A Case Study of Requirements Specification in an Agile Project

Master's thesis

Karoline Lunder

Department of Informatics

UNIVERSITY OF OSLO

2. May 2014

Abstract

Requirements specification when using agile development methods is a research area that lacks research. This is why this topic is particularly interesting to investigate. In this thesis I have explored how thorough the requirements should be specified before one starts development in an agile project. I studied a single case where a large IT-consultant company was the vendor of a big and complex exam system. The customer had not specified the requirements before the vendor company was contacted. Therefore, the requirements had to be found in cooperation. The method that was used for finding the requirements was particularly interesting to examine. The requirements were specified in an 80% level of detail to find the scope of the project before the project in itself started. The last 20% of the requirements were not decided until right before the development sprint started and were specified in a document called detailed functional specification (DFS). This document contained a detailed description of the work tasks that were supposed to be supported by the system, the IT-services that were needed for the given work tasks and screenshot(s) that explained the IT-services. The screenshots were further detailed with explanation of the functionalities the screenshot provided. This was so that developers ideally did not have to do anything more than read the description and start developing.

The first description of the requirements were in the form of user stories, but it was discovered that user stories were not covering enough, were hard for the customer to change and navigate through and it was difficult to discover dependencies between user stories due to the large size of the system. Therefore, it was decided to supplement the user stories with the DFS to end up at the right level of detail for this project. The user stories were in the DFS only used as a guide to find the IT-service that covered the user story. It was the DFS and not the user stories the developers ended up working from.

One can conclude that user stories do not provide a sufficient level of detail and is not suitable for a big project due to the amount of user stories needed for specifying all the requirements. The DFS and delaying the last 20% of detail until right before the sprint resulted in this project, in smaller changes and a project that at the point of the study was perceived as a success.

Acknowledgements

There are many people that has helped me complete my master thesis. On that account I would like to thank my fantastic supervisor Dag Sjøberg at University of Oslo for helping and guiding me through the writing of my master thesis.

Moreover, I would like to thank the vendor company and the customer for letting me study the case. In addition, Project Manager in the vendor company for helping me get in touch with relevant people in the project and assisting me in finding the correct information from the project.

I would also like to thank my mother for supporting me through all the times that my motivation was low and my father for inspiring me and motivating me to start studying informatics at the University of Oslo. He was a great inspiration when it comes to choosing this field of work.

Contents

List of tables

List of figures

1 Introduction

During the last couple of years there has been an increase in the popularity of using agile methods in software engineering. Because the rapid changes in software requirements has to be dealt with, traditional requirements engineering might not be the approach one would want to use. This is due to the rigidity of the more traditional approaches. Instead, one would want to use a requirements engineering process that is adapted to agile development. However, requirements specification when using agile development methods is a research area that has not been very thoroughly researched. This is why it is particularly interesting to find out more about this topic.

There are some claims in the agile community that you do not have to put a lot of effort into specifying the requirements before you start development. This is because agile development facilitates changing requirements and specifying them on the fly. In addition, it is claimed that user stories and tests are all the documentation you need. However, it might be seen as a bad excuse to postpone work that regardless has to be done at some point or another. Whether these claims are justified is a subject that is interesting to investigate further.

1.1 Problem formulation

The main research questions I have explored in this thesis are:

RQ1: How thorough should the requirements specification be before development starts when using agile methodologies?

RQ2: How can one make the specification thorough enough?

RQ3: When are user stories sufficient and when are other techniques or tools required?

1.2 Method

I have done a case study on a project that is run by a big Norwegian consultant company. In the chosen case, elements of an agile development methodology called Scrum were used. This project was studied over a period of 6 months where I studied the current situation of the project and information from a previous iteration of the same project.

I explored what people with different roles in the project thought about the changes that had been done when it came to the specification of requirements and their experiences with specification of requirements when using agile development in general.

This was done by doing a case study and by using triangulation, where I interviewed participants in the project, observed meetings and looked into project documentation.

1.3 My contribution

My contribution with this thesis is that I have enlighten an area where there has not been done a lot of formal research. By doing a case study, I have found that when specifying requirements in a large and important agile project it might be beneficial to specify the requirements on an overall level before the project is started and wait with the final details that do not affect the cost until right before the sprint starts. I have also found that user stories on its own do not deliver the right amount of detail, they are hard to navigate through and make changes on when the project is large and complex.

1.4 Structure of thesis

The structure of the paper is as follows: Chapter 2 introduces what agile requirements engineering is by delivering some background information about requirements, agile developments, different methodologies and go deeper into research on the subject I was studying. Chapter 3 introduces the use of methodologies, how they were used, and strength and weaknesses with chosen methodologies are elaborated. Chapter 4 introduces the case in detail with different case specific concepts, information about the project and why this specific project was chosen for the study. Chapter 5 describes the results that were found from the study. Chapter 6 discusses the results and the significance of the results from the study. Chapter 7 explores possible limitations to the study that was conducted. Chapter 8 presents the conclusion from the study. Chapter 9 expresses future work that is necessary in this field.

2 Agile requirements engineering

The requirements in the process of developing a system are very important. They are there to make sure the system that is developed meets the criteria that the system should meet.

2.1 Requirements

To express exactly what an information system should do one express the system in the form of requirements. Requirements can be defined as "the description of the services provided by the system and its operational constraints" [1]. The requirements should represent exactly what the customers need from the system. This representation is interpreted by the developers who develop the system. According to [1], the term requirement is not used consistently in the software industry. Sometimes it is used as a high-level, abstract statement of a service that the system should provide, other times it is a more detailed description of a small functionality. In other cases it can be a high level description of a constraint that is put on the system.

Requirements are generally divided into two categories: Functional requirements and non-functional requirements. We can also separate requirements into two different levels of description, user requirements and system requirements.

2.1.1 Functional requirements

According to [1], functional requirements are:

> *"Statements of services the system should provide, how the system should react to particular inputs and how the system should behave in particular situations. In some cases, the functional requirements may also explicitly state what the system should not do".*

To summarize it briefly, a functional requirement is a description of what the system should do. To illustrated this with an example: If we, for instance want to describe a concert booking system, an example functional requirement could be "The user should be able to view his or hers previous concert bookings before booking a ticket".

The functional requirements specification should according to [1] be both complete and consistent. By complete it is meant that when writing functional requirements all the services that the system should provide must be defined in the functional requirements. By consistent is meant that the requirements that are specified should not be contradictory. In large systems, it is hard to achieve this consistency. This is because different stakeholders might have different needs. The needs can also be contradictory. Consequently, writing a large specification may result in inconsistencies.

2.1.2 Non-functional requirements

According to [1], non-functional requirements are:

> *"Constraints on the services or functions offered by the system. They include timing constraints, constraints on the development process and standards. Non-functional requirements often apply to the system as a whole. They do not usually just apply to individual system features or services".*

Non-functional requirements are often a little bit more complex than functional requirements. Rather than specifying what the system should do, they specify constraints on how the system should deliver the services and functions the system shall provide. They are seldom related to an individual feature of the system, but rather represent a constraint on the system as a whole. These constraints are often concerned with security, performance, usability and other emergent system needs.

The fact that non-functional requirements specify constraints concerned with the emergent system needs makes the non-functional requirements particularly important. Failing to meet a functional requirement is possible to work around from a users point of view, but failing to meet a non-functional requirement is critical. For instance, failing to meet the security needs of a banking system might result in the system being unusable.

There are three types of non-functional requirements. Product requirements, organizational requirements and external requirements [1]. *Product requirements* specify the product behaviour [1]. *Organisational requirements* are requirements that are derived from policies and procedures in the customer's and developer's organization [1]. *External requirements* are all requirements that are derived from factors external to the system and its development process [1].

An example of a non-functional requirement concerning performance is: "The system should have a mean response time of less than 1 second". In many projects the non-functional requirements represent a problem. Because they do not represent visual results that the developers can see right away. Non-functional requirements are therefore often something that is postponed right until the end of a development sprint or for example right before the system is to be delivered or finished.

2.1.3 User requirements

According to [1] user requirements should:

> *"(...) describe the functional and non-functional requirements so that they are understandable by the system users without detailed technical knowledge".*

When user requirements are specified, they should only describe external behaviour and no system design characteristics. They should contain no use of software jargon, structured notations or formal notations [1]. These requirement do not describe how the system is to be implemented and should be expressed through a use of simple language that is possible to understand for the user.

2.1.4 System requirements

System requirements can be viewed as expanded versions of user requirements. They are often used by software engineers as the starting point for system design. The system requirements add detail to the requirements by for example explaining how the system can provide what the user requirements ask for. They describe external behaviour of system and operational constraint, not how the system is designed or implemented [1].

2.2 Requirement engineering in general

The activity of requirements specification can be described as

> *"translating the information gathered during the analysis activity into a document that defines a set of requirements" [1].*

Specifying the requirements is a part of the traditional software engineering process and one of the fundamental activities that are common to all software processes. This is an important step in the development of a software. In different methodologies the requirements specification is organized differently and can be presented in different order.

The aim of requirement engineering is according to [2] to help know what is to be built before the development starts to avoid high costs by needing to do work again. This goal is according to [2] based on the assumptions that: The later mistakes are discovered the more expensive it will be to correct them [3]. In addition the assumption that

> *"it is possible to determine a stable set of requirements before system design and implementation starts" [2].*

2.3 Agile development

Agile development is an iterative development approach that focuses on producing value for the customer. Some of the popular agile software development methods are Scrum[4-6], Kanban[7, 8] and eXtreme Programming (XP) [1, 5]. When developing with agile methodologies there is required a close collaboration with the customer to make sure that the end system is exactly what the customer needs at the given time, not what the customer thought they needed when the project was started.

The agile methodologies are according to [9] based on the principles of lean production. This means they are particularly focused on delivering value for the customer and having satisfied customers that appreciate the system that is delivered [9].

2.3.1 Methodologies

Agile methodologies are also used when customers are unable to provide a detailed description of the requirements for the system before development starts. The system is divided into a series of iterations. Because of this you do not need all the requirements from the start. They are invented or found on the fly. Often using prototypes to find the scope and requirements of the system.

Earlier it was believed that the best way to develop software was through careful planning and extensive documentation. These believes were challenged by the agile software development methods. It was discovered that on medium or small sized projects, there was too much overhead when using the plan based development approach. The planning sometimes completely dominated the development process and it was hard to accommodate the changing requirements using these methods. To avoid this, agile development methods were proposed. The agile methodologies are based on a set of principles, called the agile manifesto. The agile manifesto can be seen in Figure 1.

"Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan
That is, while there is value in the items on
the right, we value the items on the left more."

Figure 1: Agile manifesto [10]

The agile manifesto exists to ensure the product meets customer needs. The second statement in the agile manifesto describes how working software has more value than thorough documentation. This presents the dilemma of how thorough the requirements has to be specified when using agile development.

The principles of agile development are: *Customer involvement, incremental delivery, people not process, embrace change and maintain simplicity*[1].

2.3.2 Different agile methodologies

In agile development, there are several different methodologies. I will restrict myself to describing the methodologies XP, Scrum and Kanban. Scrum, because it is a popular agile methodology and XP because this according to [1] is one of the best known agile methodologies.

2.3.2.1 *XP*

XP is an agile development methodology that according to [1] is the best known and most widely used of the agile methodologies. This methodology was presented by Kent Beck. Good practices like customer involvement is pushed to the *extreme* when using this method. This led to the name extreme programming. The methodology also focuses heavily on programming. During development the programmers work in pairs of two. This is called pair programming. The pair programming technique is used because it is thought that two programmers produce higher quality code than one programmer. The more experienced programmer could also teach the less experienced one.

The customers are closely involved in the process of specifying the system requirements. The requirements are put together in cooperation with the customer and developer as story cards that captures the customers needs. The requirements in XP are presented in the form of user stories that are broken down into tasks, included in the release plan, implemented, integrated and tested, released and evaluated. This is expressed in the XP release cycle, see Figure 2.

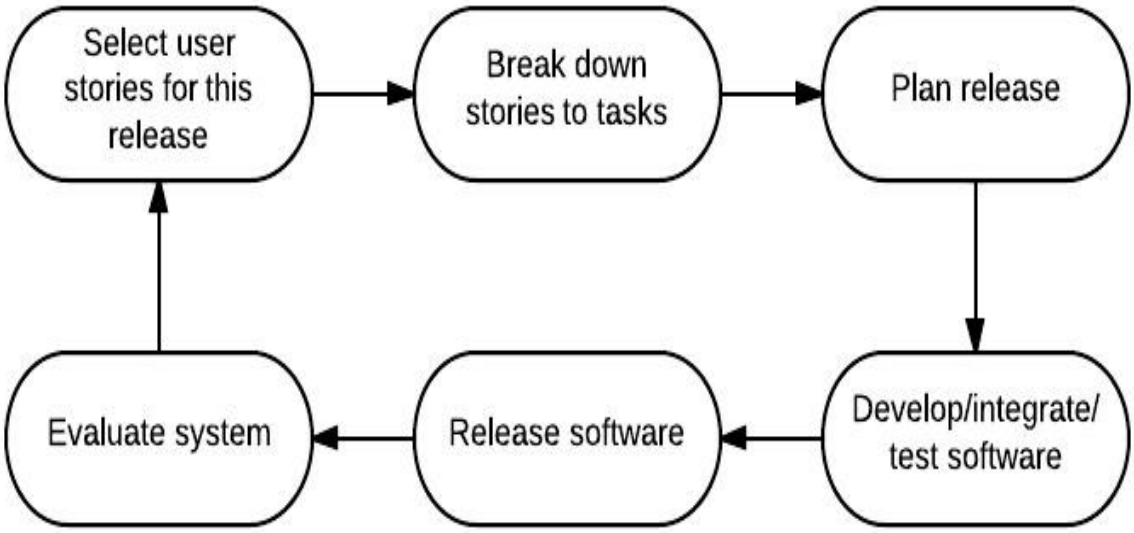

Figure 2: Extreme programming release cycle [1]

2.3.2.2 *Scrum*

Scrum is an agile system development methodology that is gaining more and more popularity. Its goal is to deliver a product during a series of sprints.

In scrum, there are three roles:

- Product owner – writes product backlog and plan sprints
- Scrum master – lead Scrum meetings and function as team leader
- Scrum team – developer

In Scrum, you have a product backlog with the different functionalities that is wanted in the finished product. The product backlog is put together by the product owner. The functionalities that the system should contain are put together in cooperation with the customer. From the product backlog, you choose tasks that are put into the sprint backlog. The sprint backlog is a collection of the tasks that are supposed to be completed during the sprint.

One sprint is an iteration in Scrum. A sprint usually lasts for 1-4 weeks. During this time all the tasks in the product backlog must be completed. The work day starts with a daily standup where the developers tells what was done the previous day, possible difficulties and what is supposed to be done today. The developer choose tasks from the scrum board which is a visual representation of the sprint backlog. One task should be small enough to be completed during one day. Ideally, one sprint should result in the completion of a shippable part of the system, but this will not always be the case.

2.3.2.3 *Kanban*

Kanban is based on the Toyota production system and lean software development. Lean software development is a set of principles derived from the Toyota assembler lines. The principles are: *optimize the whole, eliminate waste, build quality in, learn constantly, deliver fast, engage everyone, and keep getting better [12].*

In Kanban you have a maximum limit for the number of ongoing tasks. As soon as the number of ongoing tasks is below max, you can start a new one. The limit exists to avoid bottlenecks in the development process, which delay the completion of tasks. The start of a new task must wait if the maximum number of ongoing tasks has been reached.

An example of a kanban board can be seen in Figure 3.

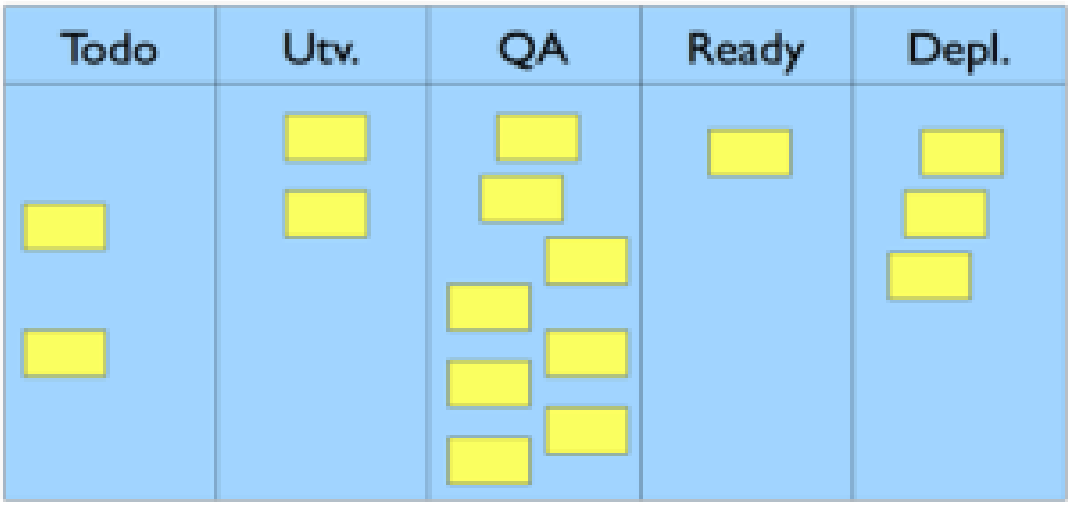

Figure 3: Kanban board [13]

The Kanban board is quite similar to the scrum board. The difference is that there is a limit to started tasks. In the Kanban board, you can see that there are tasks that are to be done, tasks under development, tasks that are in the process of QA, tasks that are ready and tasks that are being deployed.

2.4 Agile requirement engineering

One can say that the main differences between agile requirements engineering and more traditional requirements engineering is that agile requirements engineering allows and welcomes changes to the requirements at any point in the development process. Traditional requirements engineering on the other hand gathers and specifies requirements before development starts. According to [14], because continuously changing requirements are normal due to the dynamic nature of most organisation, it is difficult to specify complete and accurate requirements up front. Therefore, agile requirements engineering can be seen as a response to the rigid approach in traditional requirements engineering where all of the requirements had to be specified in advance.

Agile requirements engineering allow requirements to emerge during development and facilitate changing requirements. This is particularly convenient when all the requirements are not known from the start. According to [15], there are four assumptions underlying the agile methodologies that apply to requirements engineering. These are:

- Requirements evolve due to changes in technology, customer needs and business domains. The requirements therefore emerge during the development process instead of spending a lot of time eliciting requirements in the beginning.
- Extensive documentation not necessary
- Customers are available for frequent interaction with the developers

- Changes to the system do not cost dramatically more over time. Because evolving requirements do not increase the cost of development you can do requirement engineering iteratively

Because the requirements evolve during the course of the development, agile does not see the purpose of deciding on the requirements in the beginning and making extensive documentation that might be redundant after a little while. Extensive documentation is according to [2] seen as infeasible or not cost effective, but it is seen that agile methods tend to produce too little documentation and traditional methods on the other hand tend to produce too much documentation. Another important assumption that apply to requirements when using agile methodologies is that the customer should be available so that the requirements always are updated to match what the customer wants. Changes to requirements or the system can also be done iteratively or incrementally.

Agile approaches to requirements engineering can also be used. This is not necessarily when using an agile development method. It was confirmed in [16] that agile techniques could be successfully transferred to flexible requirements engineering processes. These findings were also supported by [17] where specifically interaction with stakeholders to help identify correct requirements were promoted.

2.4.1 User stories

When using agile development methods, requirements are often expressed as user stories. A user story is a simple way to describe what a system should do for a given role and why it should do it. In other words a short description of a single functionality that has to be implemented by the system. User stories can be used to express both functional and non-functional requirements.

The user story template is on the form "As a <user role>, I can <activity> so that <business value>" [18]. An example of a user story can be: As a system administrator, I can add users to the system so that the system will get more users. Here "system administrator" is the role, "add users to the system" the activity and "that the system can get more users" the business value.

The user stories are implemented into the project as a series of tasks. The concept of user stories was originally used in XP. At the moment it is one of the most used methods for expressing requirements in projects where agile development methodologies are used. The user story is then supplemented by a test case that covers the acceptance criteria for the given user story. In agile this is seen as sufficient, but the use of only user stories to express requirements are according to the experience of [19] only partial successful. The type of system and domain should according to [19] dictate if only user stories should be used.

When there are too many user stories because of big and complex projects. It might be difficult to navigate through and keep record of all the different user stories. Therefore it might be a good idea to group user stories or at least connect them in a logical manner.

2.4.1.1 *Grouping of user stories*

When a big project is being developed. A good idea might be to group the user stories. This is because a project with hundreds of stories can be difficult to manage, might be duplicated or contain similar user stories without one noticing it. Changing the requirements expressed as user stories might also be difficult due to unmanageable amounts of stories. Therefore, one might want to group the stories when there are many of them.

2.4.2 Detailed requirements when using agile?

A study [16] has tried to add the following two criteria to decide if detailed requirements engineering will add value to the development of a functional area. They ask:

Is the business risk high without good requirements? Moreover, *is the business risk lower with good requirements?* These questions might seem trivial, but when for example experienced developers with domain knowledge are involved in the development, one might answer the first question with "no". In addition, it might be very difficult to find defects in textual and visual description of a functionality. Therefore, the business risk might not be lower, even though the requirements are thoroughly specified. This again led to not needing a thorough or complete specification. A complete specification is defined by [16] as: all functional areas of a product being described, and all are described at a uniform level of detail.

2.4.3 Non-functional requirements when using agile

According to [2], the handling of non-functional requirements are not well defined when agile development methodologies are used. This is because the customers or users are the ones who talk about what they want the system to do. They usually do not think about the non-functional requirements. This might lead to problems during development because of the need to think about the non-functional requirements when the system is being developed. Therefore, [2] states that agile methods need to include more explicitly how the non-functional requirements should be handled.

2.4.4 Requirement changes and project success

Requirement changes can have a huge impact on the success of a project. Late changes to the requirements tend to affect the development of the system negatively. It was found in [20] that agile practices can help moderate the negative effects from requirements changes. The agile qualities like product adaptability, efficient execution and willingness to change had a positive effect on the relationship between requirement change and project success. However,

qualities like iterative development, work climate and continuous customer integration did not reduce the negative effect of changing requirements [20].

2.4.5 Benefits and side-effects of agile practices

In [21], five requirements engineering related agile practices were introduced in a company. It was shown that using agile practices addressed some of the requirements engineering challenges like over scoping and bad communication (communication gaps/lack of communication), but agile practices also introduced some new challenges like not being able to ensure sufficient competence. A study[15] identified six agile practices which were:

1. Face-to-face communication over written specifications
2. Iterative requirements engineering
3. Requirement prioritization goes extreme
4. Managing requirements change through constant planning
5. Prototyping
6. Use review meetings and acceptance tests

Another study [22] identified the same agile practices plus an additional practice:

7. TDD

It was confirmed by [23] that matching features of modest sized information system development and agile requirement engineering practices worked well with modest sized information system projects. This was confirmed by applying face to face communication, iterative requirement engineering, rapidly changing requirement priorities, inspective meetings and continuous plans to manage requirement changes to a project and observing success.

2.4.6 Challenges

There has according to [15] been encountered several challenges when finding requirements in agile projects. These are pointed out here:

1. Problems with cost and schedule estimation
2. Inadequate or inappropriate architecture
3. Neglect of non-functional requirements
4. Customer access and participation
5. Prioritization on a single dimension
6. Inadequate requirements verification
7. Minimal documentation

3 Methodology

I conducted a case study to get a better understanding of how you can and should do requirements specification in a real world agile project. The choice of study has advantages and disadvantages depending on what type of research question you choose. In this chapter I will explain the choice of methodology both for finding relevant articles for my thesis and conducting the study.

3.1 Information searching

In my literature search I searched through IEEE, springerLink and wiley online library. This literature search started as a structured search where I found some articles that were relevant for describing and finding background information on requirements engineering when using agile practices. In addition, how requirements are found and different challenges that can be met. The search was performed to find out if there had been done any research on the topic of how detailed the requirements specification when using agile development should be. More specific if there had been done any research on when to use user stories and if user stories by itself is enough and when it might be enough. It proved quite difficult to find out more about the topic due to the lack of similar research.

The search that was made was through document titles that contained the string: *agile AND (specification OR engineering) AND (requirement OR requirements).* The search did not return many relevant articles. Therefore, I supplemented the search by going through some of the articles that were referenced in the articles that were found by the search string. This resulted in a couple of more articles.

3.1.1 Strengths and weaknesses with the literature search

I could have chosen to search through several different online libraries to find relevant literature, but the ones that were chosen were some of the most popular online libraries when it comes to scientific literature. Therefore, I assume that the most relevant articles related to my case were to be found there.

Another weakness could be my search string. To avoid missing relevant articles I tried different search strings to find one that captured the most relevant articles when it came to research related to my research question. I chose the string that returned the highest amount of relevant articles.

Some of the articles were found through references from articles found in the literature search. This was to avoid missing relevant articles. The literature search in itself did not return many relevant articles due to lack of research on the topic.

3.2 Case study methodology

When doing case studies you can choose to do a single-case study or a multiple-case study. Multiple case studies are when the same study consists of more than one case and single case studies are when the study consists of only one case. There are two variants of single-case studies, holistic and embedded. Choosing a single case study can be justified when certain conditions are fulfilled. The case has to represent:

> *"(a) a critical test of existing theory, (b) a rare or unique circumstance, or (c) a representative or typical case, or where the case serves a (d) revelatory or (e) longitudinal purpose." [24].*

3.2.1 Choice of methodology

There are several reasons for why I chose a case study to find out about my problem. Case studies are according to Yin [24] preferred when the research question has the form "how" or "why". Because my main problem formulation starts with "how", a case study seemed like a good choice. The use of case studies are also preferred when the researcher has little or no control over the events that are studied and when the focus according to [24] is on a contemporary phenomenon in a real-life context.

The choice of a single case study can be justified because the condition (b) a rare or unique circumstance is fulfilled. The case was also chosen because of its obvious relevance. More cases would have been desirable, but because relevant projects are not easy to come about I found only this one case. This makes the study a single case study. There are disadvantages with only having one case which is discussed later in the thesis. I justify the choice of only having one case because I have found a case that is suitable for my particular research question.

3.2.2 How the case study is conducted

The gathered data was chosen on the base of availability. It focuses on one project divided into three smaller projects. I was fortunate enough to be able to interview six people, have several talks with the project manager, attend four meetings, obtain company, and project specific documentation that I could look into and analyze.

There are according to [24] three principles of collecting data when doing case studies. These are: *Use multiple sources of evidence*, *maintain a chain of evidence* and *create a case study database*.

These principles have been covered by:

- Using triangulation when analyzing data: Interviews, direct observation and documentation

- Chain of evidence has been followed by citing important interviews and observations that lead to the conclusion
- There has been created a case study database, but the documentation, interviews and observation notes are not available for others due to confidentiality

Evidence in a case study can come from many sources. This range from amongst other documentation, archival records, interviews, direct observation, participant observation, and physical artifacts [24]. The methods chosen in this study is interviews, documentation and direct observation. To find results that are more valid, I used triangulation. According to [24] a good case study uses as many sources as possible.

3.2.2.1 *Interview*

According to [24], one of the most important sources of evidence in case studies are interviews. This is why interviews are the primary source of evidence in this study.

Some of the strengths of interviews are according to [24] that you can focus the interviews directly on the topic of the case study and it provides insight into the opinions of the subjects. There are also some weaknesses to this type of source. Amongst other that it can be inaccurate due to poor recollection of the interview. To avoid this, I have chosen to record every interview with my phone so that important information is not overlooked. Other problems with this type of source is that there might be bias due to poorly articulated questions and responses.

I interviewed participants in the project that possessed different roles. To map and get an understanding of the project I had three talks with the project manager from the vendor. The project manager gave me material and information about the project so that I could understand the project and the processes that were used. He also mediated contact with the people I interviewed for my study. To secure that I had no big misunderstandings when it came to the case, he also helped me review what was written about the case and context. Therefore, the project manager worked as a key informant when it came to the case. In addition, he worked as one of the interviewees.

The chosen interview form was semi-structured interviews. The reason semi-structured interviews were chosen was that the interview method made it possible to ask questions like: “tell me about this and this process.” In addition, it allowed the interview to be more conversational than if a structured interview had been chosen. Because I needed to gather information about how they performed their processes and how they perceived these processes, semi-structured interviews were a suitable choice.

To avoid missing relevant information from the interviews, I recorded the interviews with my phone. After the interviews, I transcribed them and grouped the answers into different categories.

I managed to get interviews with six people who possessed central roles in the project. Their roles are described in Table 1. This table describes their role and the responsibilities that belong to given role.

Table 1: Roles and description

Role	Description	Responsibilities
Customer	Project manager from the customer	•Product queue and elements •Delivering what vendor needs •Reporting to steering committee •Make sure system is what they want
Functional architect OFS	Functional Architect Analysis and design next iteration	•Plan scope for the system before development is started. •High-level design which goes into the contract and forms baseline product back-log. • Pre-project activity that relates to traditional analysis and design phase.
Functional architect DFS	Functional architect current iteration	•Responsible for DFS and specifying the requirements •Lower level design/specification that breaks the product backlog items to components ready for development •Is prepared during construction phase of the project.
Functional responsible	Shares responsibility with functional architect DFS	•Manages functional requirements •System is what customer wants •Guide customer Link between technical and user
Scrum master	Scrum master. And 50% developer	•Facilitates development
Project manager	Project manager from the delivery side	•Team, •Delivery •Contract •Time, •Scope and quality •Cost

The interviews were held in Norwegian. This means that all the quotes from the interviews have been translated to English for the purpose of the thesis.

After discovering some uncertainties around some of the answers from the interview subjects. I sent a follow up mail to get clarifications.

3.2.2.2 *Look into documentation*

Documentation is a source of evidence that can be on many forms. Some of the strength of this type of source is according to [24] that it can be reviewed several times and it is not created as a result of the case study.

I got a hold of the OFS from iteration 2 and got the opportunity to study how this looked like. There exists no similar documents as the OFS from iteration 1 because OFS was something that was introduced during the second iteration of the project due to the need for a better documentation of the scope of the system.

Reviewed documentation was:

- High-level functional specification (OFS) from iteration 2
- 2 detailed functional specification (DFS) from sprints in iteration 1 and iteration 2
- Vendor explanation of integrated architecture framework (IAF)

To get a better understanding of the case I first got the OFS from iteration 2 from the project manager. This gave me a good overview of how they specified the system. I found out that I needed some more information and therefore got the DFS from one sprint in iteration 2 and one sprint in iteration 1. To understand how the framework they used worked, I got the company specific description of IAF. This was so that I could get a better understanding of how they worked and supplement what was said in the interviews. I also wanted to see whether the documentation really matched what they said in the interviews.

3.2.2.3 *Direct observation*

Direct observations is a form of source that might be useful to provide additional information about a topic being studied [24]. The strengths when it comes to observations are according to [24] that observations cover events in real time and the context of the case. Some weaknesses might be that observations are time consuming and when under observation the event can happen differently because someone is observing the event. The reasons for doing direct observations are to get an understanding of how they work in real life, see how their processes work and if they work as expected.

When studying my case I had the opportunity to attend four slightly different meetings significant to the case. These meetings were from different iterations and sprints in the project. The first meeting was in the process of defining the OFS for the next iteration (iteration 3). The rest of the meetings were work meetings from two different sprints in iteration 2 of the system and part of defining the DFS. Which meetings were attended can be viewed in Table 2.

Table 2: Attended meetings

Date	**19.nov**	**20.nov**	**2.dec**	**9.dec**
Type of meeting	Meeting to decide on one of the capabilities the customer wanted in the iteration and to be included in the OFS.	Work meeting DFS Last work meeting	First work meeting for DFS	Last work meeting for DFS
Agenda	Map details when it comes to a process. Agree how a process should be in the system. Agree on what has to be standardized and what the system should contain.	Look into screenshots and get a final ok from customer.	Look at screenshot sketches and approve or change. Clarify uncertainties when it comes to process	Look into changes and corrections from last work meeting Show the changes to the screenshots. There had been done a substantial amount of changes. Show some new screenshots
Iteration	3	2	2	2

3.2.1 Strengths and weaknesses when using case studies

When doing studies there are often several methods that can be used for the same problem. However, one has to be chosen. This can be based on the experience of the one doing the study. It is easier and sometimes more beneficial to choose a method one is familiar with. This is the first time I have done any scientific studies. Therefore, I chose the method that seemed most suitable for my problem. A case study is not necessarily the only choice of methodology, but due to the research questions being studied it seemed suitable. There are strengths and weaknesses when using the case study methodology. A case study is often chosen when there according to [24] are more interesting variables than "data point". This is why using more sources of evidence is a widely used strategy when conducting case studies. To achieve more sources of evidence you can use a triangulation technique. Triangulation means that you have several sources of evidence in your study. This makes the results from your study more reliable.

3.2.1.1 *Possible disadvantages of single case studies*

It is often beneficial to have more than one case when doing case studies. This is because by obtaining more sources of evidence, you have more to back up your research results. Another potential danger by choosing only one case is that it might not turn out as you thought it would. It might not cover the questions you originally asked, so locking in on one case too early is not necessarily a good idea.

3.3 Descriptive models

For representing the processes that were performed to make the requirements specification, I chose to represent them by using a model.

> *"A model is an abstract and simplifying representation of an object or phenomenon of the real world" [25].*

This was chosen because it makes it easier to get an overview of how things have been done in the project. The processes have been represented by process models which is: "A software process model (short: process model) is a model of a software process" [25]. The software process model was made using product-flow notation. This is used to make descriptive process models [26]. Product-flow notation was used because the method gives a simple description of the processes in a simple and understandable way. The shapes that were used are explained in Figure 4.

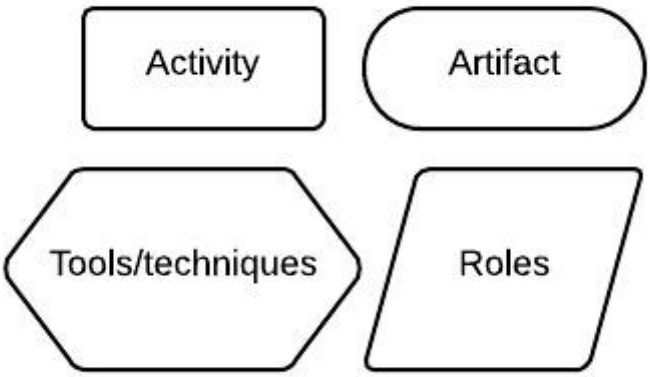

Figure 4: Explanation of shapes

The *activities* describe what is being done. The *artefacts* describe what is being used or produced by the activity. The *role* describes who are involved in the activity and the *tools/techniques* are the tools and techniques that are used by the roles in the activities.

4 Case and context

The company in which I conducted my case study is a big international IT company. They work with consulting, outsourcing and IT, where they amongst other work with creating and delivering technology. They have well over 100 000 employees.

The project my thesis focused on had adopted the scrum methodology to some degree. What is most interesting about this case is that they had done some changes in how they described the requirements during the project. The changes were done based on the experience from one part of the project to the next one. Therefore, the case is very interesting to study.

I have based the case and context chapter on some of the information retrieved by studying the documentation I have been able to obtain. It is also based on some of the information retrieved from talking to the project manager, the interviews that were held and my experiences from observing the meetings.

4.1 The system to be developed

The system that was developed is a system for digitalizing exams in schools. The main motivation behind this digitalization was that during paper exams there were enormous amounts of paper being produced. Digitalizing exams eliminates the need for paper during exams and makes it easier to correct the exams.

When the project started, no similar systems had previously been developed. It was searched thoroughly to find out if there were any similar projects, but none was found. This makes this project a pioneer project.

4.2 Description of case-related concepts

Some company-specific frameworks and methods that were used in the project are described in Table 3.

Table 3: Description of case specific terms

Concept	Description
IAF	Integrated architecture framework • Vendor specific architecture framework • Contains processes, tools and techniques to shape architectures • Used to standardize architecture • Tries to define: why, what, how and with what the system is developed • Helps define the requirements
OFS	High-level functional specification • Vendor "invented" the OFS for this case • Semi detailed description of functional specification • Level of detail: 80% • Defines all the IT-services (without details)
DFS	Detailed functional specification • Vendor "invented" the DFS for this case • Final detailed description of requirements • Level of detail: 100% • Explains work tasks, IT-services, screenshots and details the screenshots
Work tasks	• Activity conducted by specific role • Has clearly defined goal
IT-services	• A service that is provided by the system • Handles (in this case) one or more user stories
Screenshot	• The functional description of the IT-service is described from the screenshots • A detailed picture of how the screen should look like • Is further detailed by describing functionality of screenshot

These terms and what they were in the project is further elaborated in the following sub-sections to get a clearer understanding of how they were used in the project.

4.2.1 IAF

IAF is a framework for architecture developed by the vendor of the case. It is a roadmap that contains tools, techniques and processes to shape the architecture of a system. The framework is used to arrive at a common understanding of why, what, how and with what to create the architecture of a system [27]. The particular solution one arrives at by using this framework is not necessarily the best and only option for the architecture, but the company believed it is a good and practical option that can be used during the whole process of making the system.

The IAF components that were particularly used in the studied project to produce a description of the functional aspects of the system were; IT-services and work tasks. The IT-services are used to support activities in a work task as seen in Figure 5. The IT-service is further specified by a screenshot and a detailed description of the different functionalities the screenshot provided and how it is or should be provided.

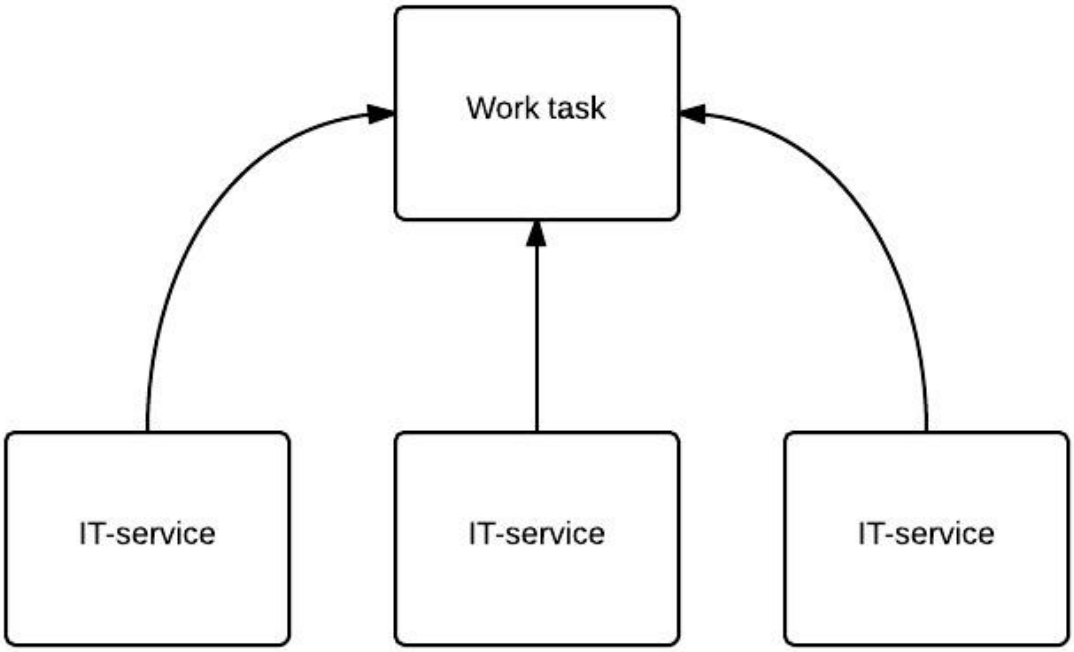

Figure 5: Requirements in IAF

A work task is an activity that is conducted by a specific role and has a clearly defined goal and result. In addition, what event triggered the work task has to be included. The activity is described on a business level and is not necessarily very detailed.

All the work tasks are divided into IT-services. IT-services are all the little things that has to be supported so that a person can do his or hers work task. The work tasks are described by several IT-services. These are the building blocks of the architecture. The IT-services describe the behavior, the service that is provided and in the case that was studied; covered one or more user stories. To provide further information, the IT-service contains a detailed screenshot, which is the visual description of how the end service should look like. To supplement the screenshot there is an even more detailed description of all the functionalities the screenshot provides and the basics of how it is provided. This description is detailed so that the developer ideally does not have to ask any questions during development.

4.2.2 OFS

The OFS is a document that contained the overall description of the system for one iteration of the project. The Term OFS was made for the particular project and was not something they had used in previous projects. Level of detail it should contain was 80%; the rest was specified in the DFS. It did not contain screenshots or long descriptions of the services that were to be provided. However, the work tasks and IT-services for the iteration were described in an overall manner so that it was possible to define the scope of the system at an early stage

in the process. The description contained the role, whether there should be a screenshot or not, if the service should be automatic, the goal of the service and what the service should provide.

The OFS was developed in cooperation with the customer in a series of meetings where the processes and functionalities were discussed with the relevant stakeholders.

4.2.3 DFS

The DFS is a document that contained a detailed description of the functionalities connected to one sprint. The term DFS was made for this particular project and was not something that was used in previous projects. Each sprint required one DFS and the remaining 20% of the functionalities were specified in this document. In a project and sprint context, development would not begin until the DFS was agreed on and verified between the vendor and the customer. The DFS contained a more thorough specification of the requirements and IT-services expressed in the OFS. These requirements were described in detail with screenshots and the user stories that each IT-service covered. The IT-service represents the final description. If the functionality described by the user story was not included in the IT-service, it had been decided that it was not necessary to implement the user story in the system. In other words, the user stories were not considered to have the same importance as the IT-services in the document. They were in practice only used as a reference to find out which IT-service covered the specific user story.

4.3 General information about the project

The project was big, in both complexity and size. The customer had a somewhat basic idea of what they wanted, but the requirements were not known beforehand. To find the requirements it was hired a consultant company to help them figure out what was wanted from the system. Once the customer had decided what they wanted, they needed help to either find a system that satisfied what they wanted, or build a new system entirely from scratch. This was done in something they called a preliminary project.

4.3.1 Divided project into smaller projects

It was discovered that making the wanted system would be a big assignment. Therefore, it was decided to divide the project into smaller pieces, so that it would be more manageable to develop. The project was divided into 4 individual lanes, where 3 of the lanes were sub-projects called iterations (iteration 1, iteration 2, and iteration 3) and where one lane was called analysis and design. They arrived at the solution described in Figure 6.

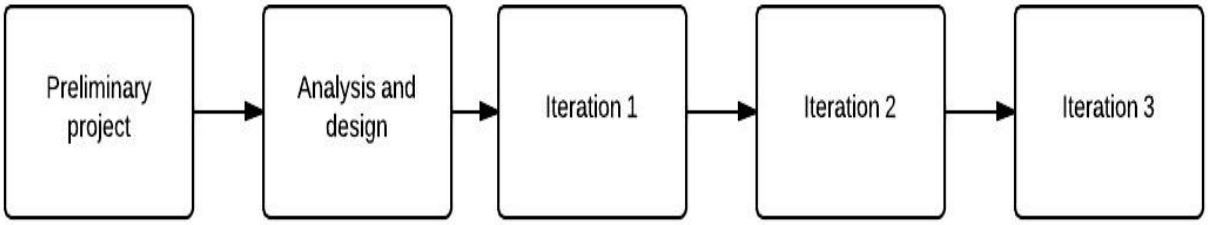

Figure 6: Course of the project

4.3.2 Preliminary project

The preliminary project had three questions to address:

1. Buy or develop
2. If develop, what technology (i.e. .NET, JAVA, etc.)
3. How to realize the project, i.e. project model, co-operation with vendors, type of contract, etc.

The preliminary project was started to find out what technology to use and if the system was to be developed or bought. The vendor searched thoroughly for similar projects in both Norway and abroad. The conclusion was that no one had done anything similar to the solution that was wanted. Therefore, it was decided that the solution had to be built from scratch. This meant that the project was a pilot project and that the developers had no similar previous projects where they could borrow inspiration. This resulted in the need of a proof of concept. The proof of concept became sort of an initial mapping of the requirements. There were five architects on the job of mapping different areas of the system. Because of pilot work and high risk, they ended up using a contract with running hours.

In the preliminary project, there was close cooperation with the end user, which was exam sensors, academic and others involved in the use of the application. This was to find out what was really needed. The result of this preliminary project was an analysis of the situation and how the desired future situation looked like. Which again led to the initiation of an analysis and design project.

4.3.3 Analysis and design

The customer had a somewhat basic idea of what they needed from the system, but the requirements were not determined. Because of the lack of knowledge when it came to the requirements, some big questions emerged. Amongst other, they wondered how they could solve the estimation for the project when there were no requirements. This was one of the big questions and lead to the decision of having a thorough analysis and design phase before development started. They mapped the requirements for the tree iterations as user stories. The

customer and the vendor company cooperated on finding the requirements for the systems. This was done by using workshops where they ended up making a document that was not divided into iterations, it described the scope. Then, there was a prioritization where elements were picked and they chose what they should start with.

4.3.4 Iterations

The development of the system was divided into three big iterations. One iteration was sort of its own project. The iterations were named: Iteration 1, iteration 2 and iteration 3 as can be seen in Figure 7. For each iteration more and more of the system was finished and in the next iteration, they built on the already developed system, until the final iteration where the whole system was to be done.

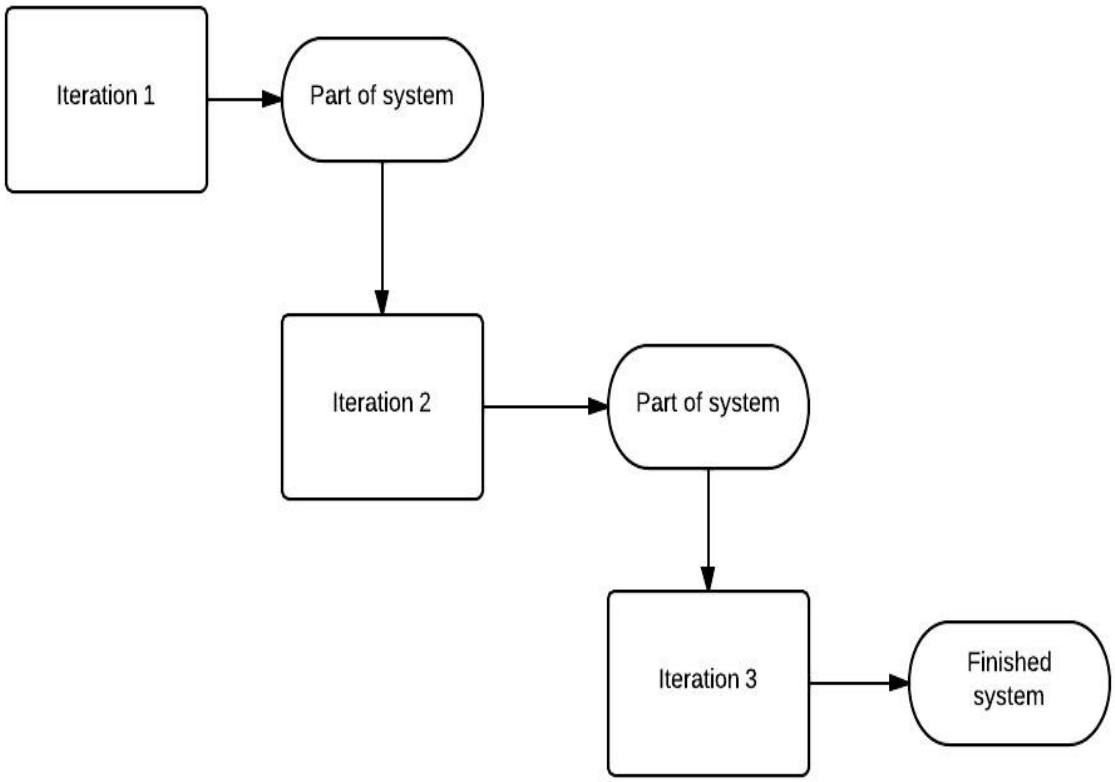

Figure 7: Overall description of iterations

The iterations were divided into several scrum sprints where the most important functionalities were implemented in iteration 1 and iteration 2. This is because the customer, at the time of the writing was not yet entirely sure if they could afford or want the third iteration.

4.3.5 Development methodology

An approach to the agile development method Scrum was used. One might call it Scrum, but it is not Scrum in the exact way Scrum is specified in the methodology. Many of the elements from Scrum were nevertheless used. However, there was used a project management method on top.

Similarities with scrum were amongst other:

- Sprints
- Artifacts, activities and most roles + a couple of extra
- Easy to adjust and do changes
- Close cooperation with customer
- Write specification continuously
- Use some of the agile principles

It was Scrum in the sense that there were sprints, artifacts, activities, roles in the development team, backlog and demos. It was also used quite agile change management where it was not hard to adjust and do changes if wanted. They had close cooperation with customer, where it was easy to ask questions if something was not clear.

However, the project was not completely agile. Differences that were pointed out were:

- No use of continuous deployment
- Decided scope before they started
- Thorough analysis and design phase
- Thorough acceptance phase
- Missed some elements
    - Scrum team not doing estimation
- Had a project manager

Continuous deployment to production was not used because it was necessary to secure that the project was what was agreed on. Unlike agile the scope was also decided before they started, but unlike waterfall, not everything was written in stone in the beginning. It was pointed out to me that this is an approach that is more or less normal in projects where you have an external developer and the project is not an "in house" project. It was also used a thorough analysis and design phase and a thorough acceptance phase. You can therefore say that a waterfall like approach to analysis and design was used, but the change management was very agile. Other differences were that a clearly defined plan for the project was used, but for each sprint, they could more or less include whatever they wanted as long as it was inside the scope that was defined. Other differences were that the Scrum team were not the ones doing the estimation on the project and they had a project manager, which according to Scrum, is not necessary.

Each scrum sprint was two weeks. The number of sprints varied from iteration to iteration. The scrum sprints were inside an iteration of the system which can be seen in Figure 8. Each iteration contained a series of scrum sprints, in which the development was conducted.

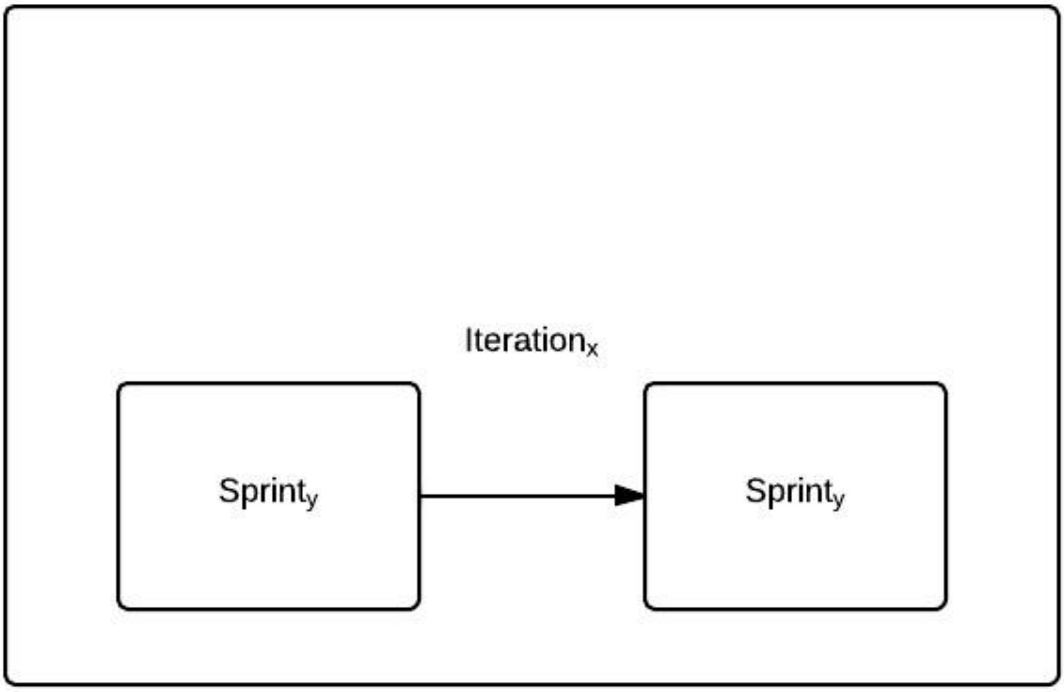

Figure 8: Scrum sprints

4.3.6 Finding requirements in project

Because the requirements were not known in advance and the customer did not have the requirements specification finished, the process of finding the requirements was a somewhat difficult one that required a huge amount of cooperation between the customer and the consultant company.

In this particular project, the collection of requirements was done differently from how they usually do it in other projects. Normally, the vendor receives a requirements specification where they in a bid phase answer how they want to solve the requirements based on the specification delivered by the customer. Then they go into the project in analysis and design where they detail and agree on the content of the requirements. This is because there is always some uncertainties where there has to be narrowed in on a final agreement where an understanding of the requirements is arrived at. This is something one does not have early in a bid phase.

In this project, there were no requirements before they started. This is particularly interesting because this is not something one usually encounters. The customer and the vendor had to develop the requirements specification together.

4.3.6.1 *Methods used for finding requirements*

To find the requirements, the vendor had five architects that went in first in the analysis and design phase. The vendor was responsible for handling and facilitating the process of gathering requirements. They sent the customer through a series of workshops in all the relevant areas, gathered the requirements and transformed them into something that was to be developed later on.

Different stakeholders from the customer were engaged in finding the requirements for the solution and the academic and administrative requirements. There were no use of specific methods or tools for defining the requirements other than: Meetings and workshops. These meetings and workshops were between the customer and the consultant company where they tried to involve all relevant stakeholders in the system. The goal of the meetings and workshops was user stories as seen in Figure 9.

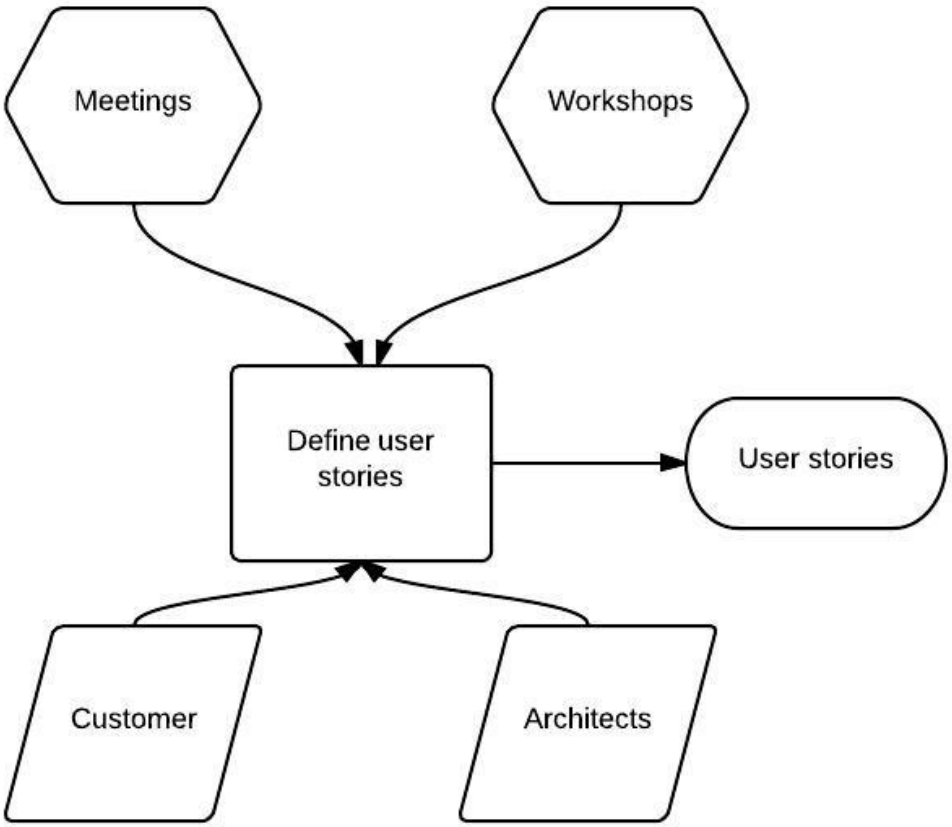

Figure 9: Defining user stories

Later it was decided that they wanted user stories and IT-services. Where IAF was used as a method for detailing what was necessary to cover the user stories.

4.3.6.2 *The process of specifying requirements*

When specifying the requirements, the starting point was a description on a very high level. This description consisted of what the system should do, should contain and the architecture of the system. This is the OFS. First, the requirements were found on an overall level in the preliminary project. These requirements were further refined during the course of the project where they started with a number of items from the customer. These items were not system requirements but more business. This was sort of a list of capabilities that the customer wished to have and that they wanted supported by the system. The capabilities were the starting point for the iteration. After this, they went deeper into the different capabilities to find out how and what the different capabilities were. When there was an agreement on how things should be done, they sketched an architecture, which was a given number of IT-services. The description might also contain screenshots and automatics, an information model and an integration overview. The integration overview told if there were other applications that needed to transfer data and needed to be linked to make this work.

As the project continued, the requirements were narrowed down and gained more detail until the final and locked description was ready in the DFS right before the sprint started. This is described in Figure 10. In this figure you can see how the overall requirements were tuned as the development of the system continued.

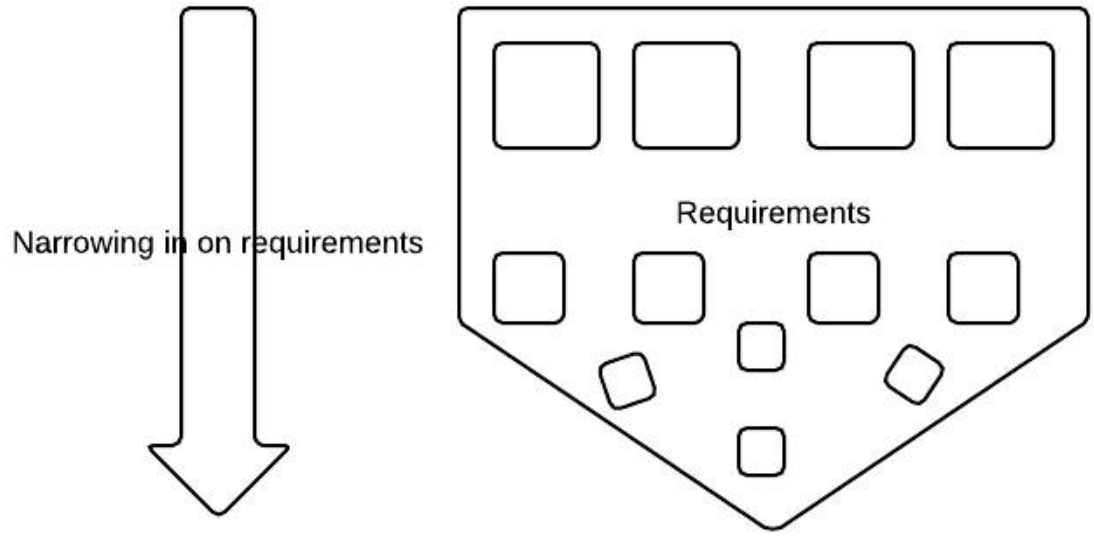

Figure 10: Tuning requirements

The OFS for the upcoming iteration was created in the current iteration (the iteration before the OFS was to be used). This is illustrated in Figure 11. Where you can see that the OFS for iteration 2 was created at the same time as iteration 1 took place. The OFS for iteration 2 was then sent into iteration 2 after the completion of iteration 1.

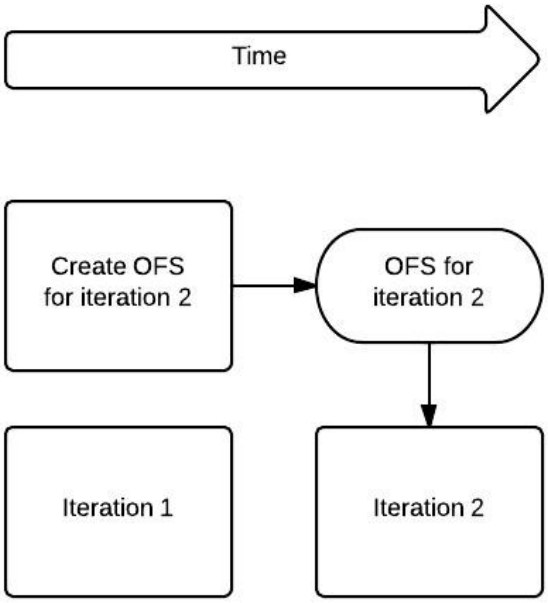

Figure 11: Creation of OFS for next iteration

The OFS was created at this point so that the customer got an opportunity to think about what they want at an early point. This gave them the chance to review and rethink whether their current processes were good enough or whether the process should be changed or defined better.

In Figure 12, you can see a descriptive overall model of how the process of refining the requirements was done for iteration 2.

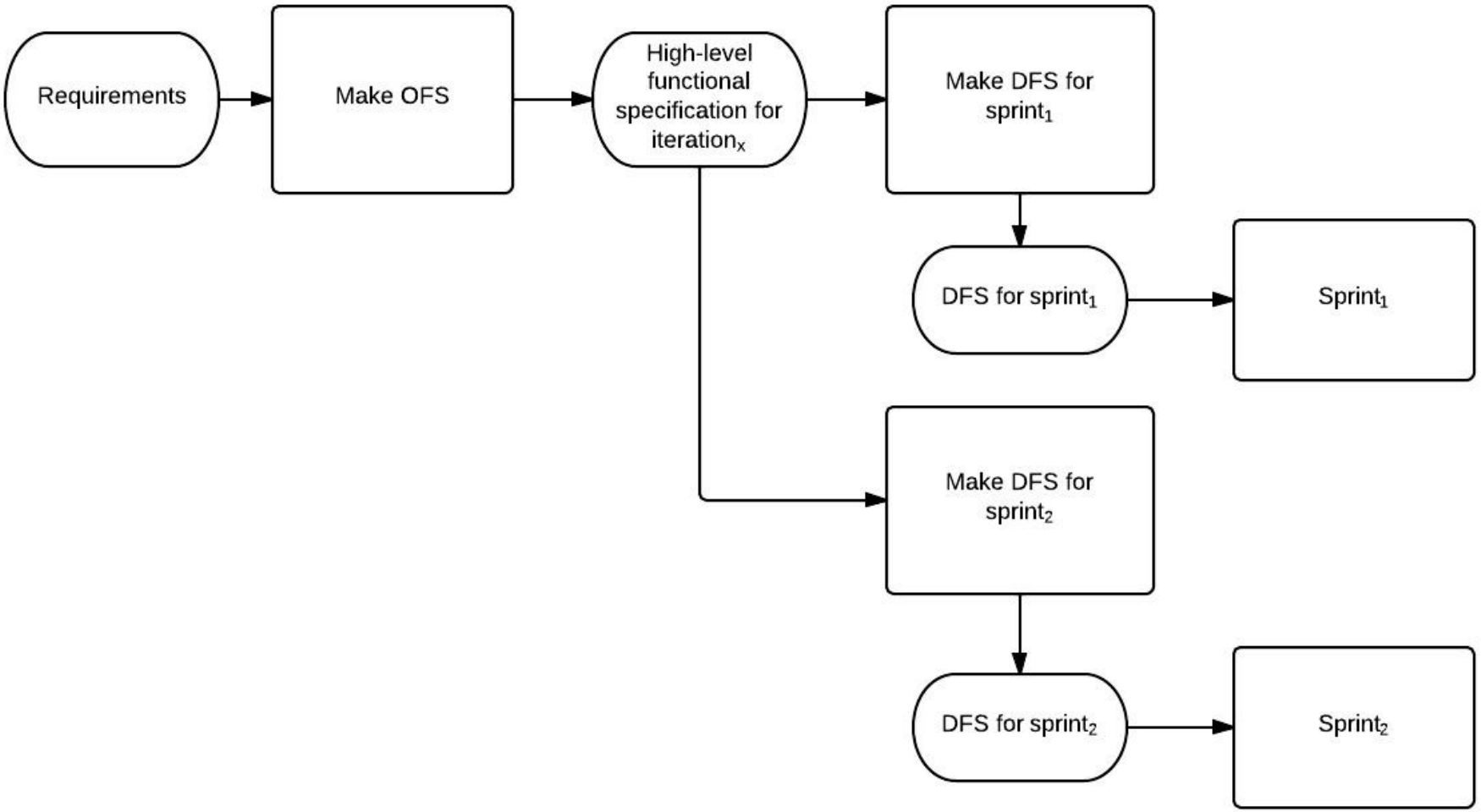

Figure 12: How the OFS and DFS is used

Requirements for given iteration were described in the OFS. These requirements were based on user stories that had been specified and were linked to IT-services. When making the OFS, the number of needed meetings was not specified before they were held. There were held meetings until the OFS was done. The OFS itself was made from a list of wishes from the customer, which had to be elaborated, put into context and written down in the OFS. This process repeated itself until the list of capabilities for given iteration were detailed in a satisfying manner so that it was possible to estimate on them. How this was done can be seen in Figure 13.

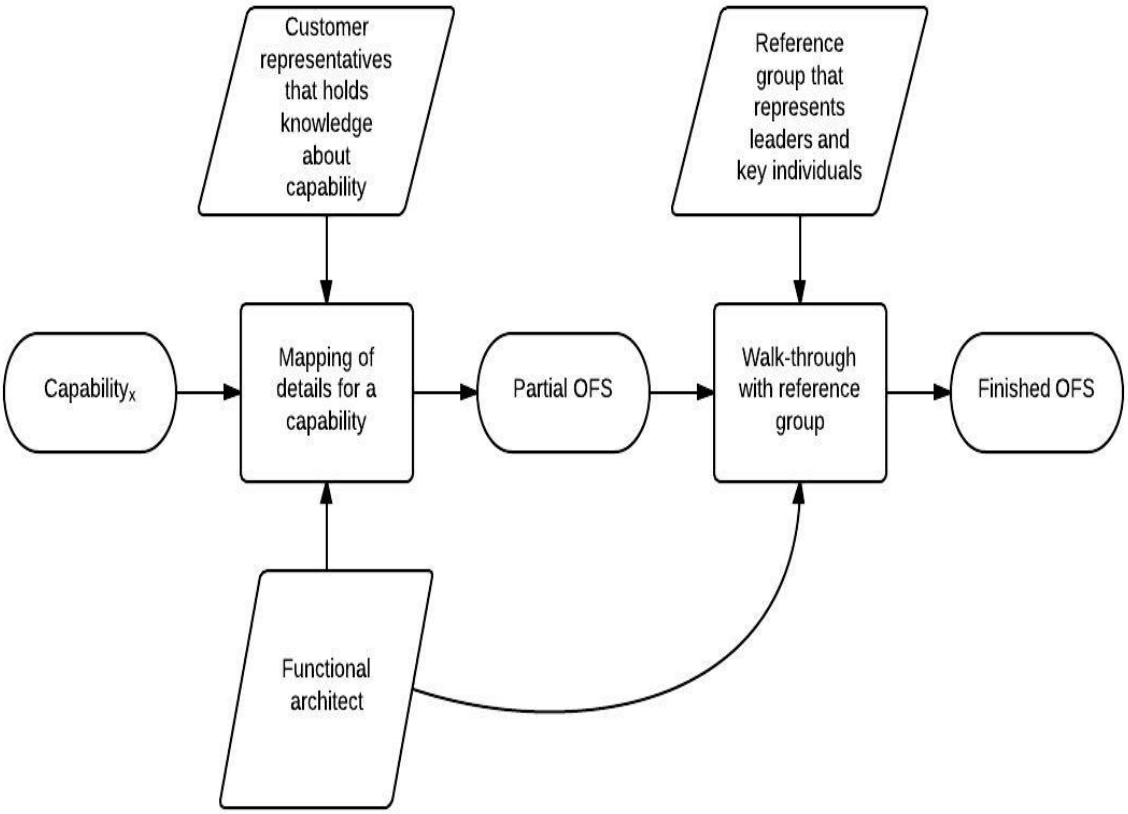

Figure 13: Making the OFS

The process in Figure 13 was for defining the architecture and business rules. As the figure shows, the capabilities for the coming iteration were discussed and re-discussed in several meetings before the OFS is finished. The roles that were involved in these meetings were the functional architect and the customer representatives that were relevant for the given capability that was discussed during the meetings.

The DFS was specified through a series of meetings and created based on the IT-services from the OFS. For each sprint there was developed a DFS where the customer prioritized which of the IT-services the sprint should contain. Details that did not impact the cost were

decided in the DFS phase. These were details like what colors were to be used on the buttons and the detailed design. The DFS phase for the next sprint could happen in parallel with the present sprint. This is so that everything would be done in time for the new sprint. The DFS phase lasted for about three weeks.

Before each sprint, there were three meetings with the customers in the process of making the DFS for the sprint. One start-up meeting and two work meetings. During these meetings with the customers, uncertainties, screenshots and more necessary details were specified in cooperation between the customer and the architects. Before the new sprint started, the DFS had to be finally approved by the customer. When the final approval was done, the DFS was locked for the sprint and the developers could start developing.

In the startup meeting, all the IT-services from the OFS related to the pending process or workflow were reviewed. It was made sure that these were still relevant. If they were no longer relevant they were eliminated. The IT-services that were still relevant were included. Guidelines on what was important for specifying them further were then created.

Before the first work meeting most of the screenshots were made. In the first work meeting the different screenshots were discussed in cooperation with the relevant stakeholders from the customer. It was decided what might work, and what might not work when it came to screenshots and what kind of information the customer really wanted. Questions with regards to the DFS were also resolved.

In the last work meeting the discussions or uncertainties from the first work meeting were resolved. The changes to the screenshots from the first work meeting were shown and discussed. After this meeting the changes were implemented in the DFS and sent for internal control by the test manager, scrum master and technical responsible (this has been implemented during the last 6 months). This was done to make sure that all the possible scenarios were implemented in the DFS and gave the test manager, scrum master and technical responsible an opportunity to know what was being implemented during the next sprint. When they had presented the feedback and it was implemented in the DFS, the DFS 0.8v was sent to the customer for approval. They gave feedback which was implemented into the DFS and a DFS 0.99v was sent to the customer for approval and final feedback was given. The DFS was then sent into the new sprint and the DFS for the sprint was locked. The final requirements specifications was therefore not finished until right before the sprint started. In parallel with the sprint, work on the new DFS was started.

Figure 14 shows how the meetings looked like.

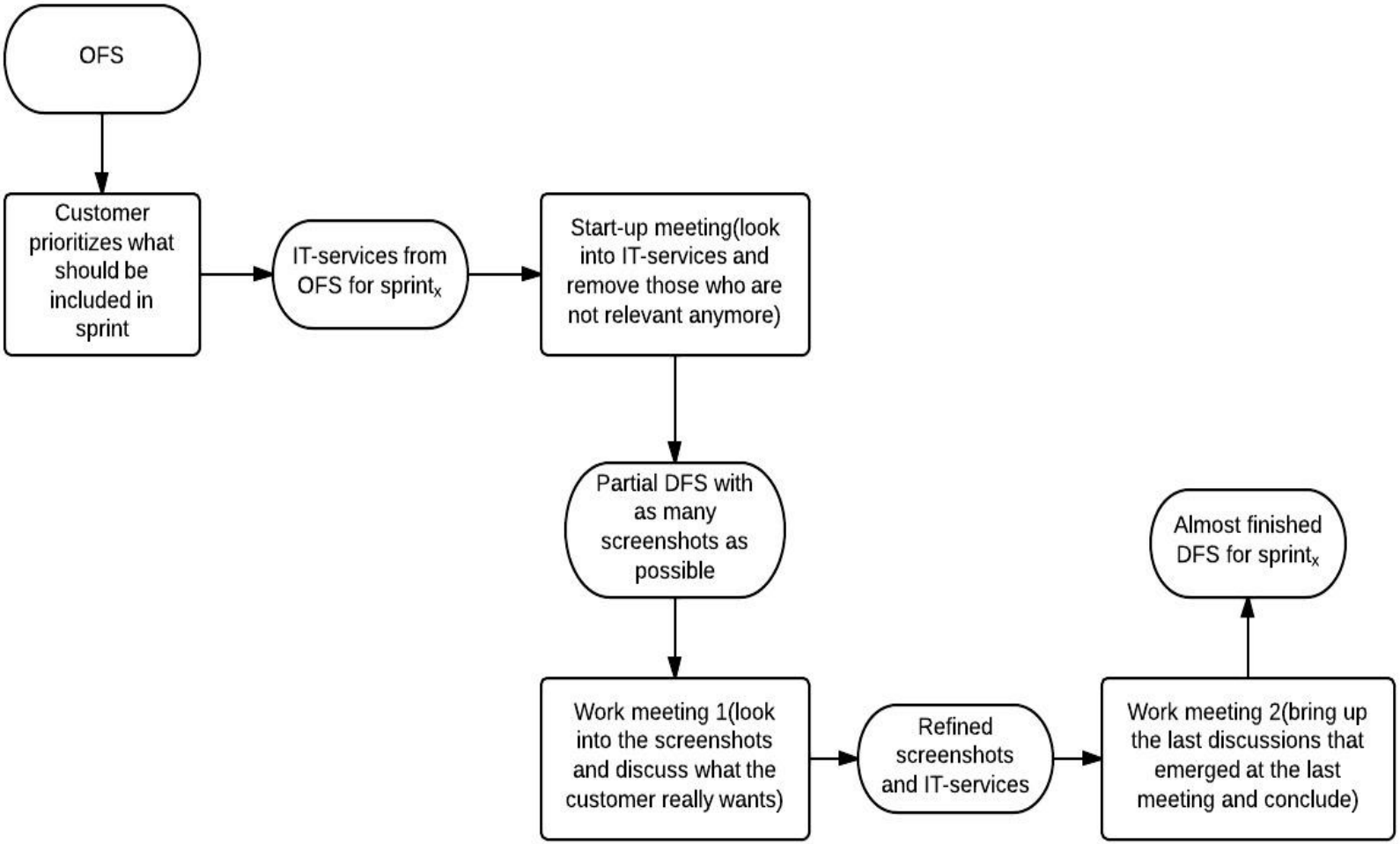

Figure 14: DFS meeting

4.3.7 Changes introduced

In the beginning, requirements were mapped as user stories, but it was seen that the user stories did not cover all the aspects of the project they wanted to cover. User stories were only fragments of what they wanted. They were also hard to manage due to the high number of user stories. It was both hard to navigate through the requirements, see the scope of the project and dependencies between functionalities. Therefore, it was decided that in the next iterations (iteration 2 and iteration 3), the requirements would be expressed as IT-services. This was decided based on the experiences of the architects. They also changed to the IAF framework for the second iteration and introduced the OFS. To summarize, they introduced these changes:

- Introduced IAF
- Introduced the OFS
- From list of user stories to something more manageable
- Before: more bottom-up, now: top-down

For iteration 1, there was created a kind of description around the user stories that was not that clear. The functional architect OFS created some more of a description around the user stories he had more information about. Where functional architect OFS, described what the customer really wanted on an overall level and a description that looked a little bit like the one that was done for iteration 2, which was the OFS. The OFS was initiated as a way to describe the overall architecture of the solution on a high level before the refining of requirements started in the DFS phase. The level of detail on the requirements did not change much; the most important aspect was that it was changed from a long list of user stories, to something more manageable.

It was also pointed out to me that the method they used before was more bottom-up, where they tried to go into more detail before developing. After changing, it was more top-down, where the level of detail was on a very superficial level and narrowed in as they went.

4.3.8 Changing requirements in project

Instead of change managing every new element that had to be added to the backlog, they collected everything and decided that if x number of elements with given estimate was no longer needed, they switched them with other elements that had to be added. This way they avoided a lot of change management. If there were more elements that had to be added than what was eliminated they had to either increase staffing, expand in time or add it to the next iteration.

It was possible for both parties to introduce change as long as there existed a mutual agreement. Smaller changes were easier to handle because they had something called smaller adjustments and changes. These smaller adjustment and changes were easy to

implement. This is because they made it possible to move within 10% of scope without it triggering a change order. In each sprint 20% of the time is reserved for this. A bigger change meant that the project needed to get more hours or one element had to be removed from the backlog. This is per definition a change in contract. This had to be approved by the steering committee.

If there was a change order it took some time to detail it. Because the detailing of the next sprint was already started, this might result in the change not being implemented before the sprint after the current one, but it depended on the size of the changes. The change has to be approved by both project managers (customer and vendor).

When it is sprint planning with the developers they sometimes proposed some new ideas. Usually big changes were not included from this, but if they introduce some smart small ideas, they might be passed on as ideas to the customer. If the customer thought that it was ok, the changes might be included. To help with prioritizing proposed changes they introduced a system where the customer was able to prioritize the proposed changes. This made it easier to see if some of the proposed changes could be discarded right away because of low priority.

4.4 Functional requirements

The functional requirements for the system were handled thoroughly by the OFS and the DFS. These requirements are the ones that are particularly discussed in this thesis.

4.5 Non-functional requirements

The non-functional requirements were specified in the initial analysis and design phase. This was for the whole solution. This means that not all of the non-functional requirements were relevant for all the iterations of the project. Therefore there was initiated a phase called detailed technical specification (DTS). In this phase, the technical responsible with the customer and the vendor agreed on the specification.

Everyone had to agree on which of the non-functional requirements were relevant for the given iteration before work was started. To make sure the non-functional requirements were handled to some degree, there was done iterative capacity testing after each sprint from iteration 2. If the capacity was not good enough because of for example too much communication between different parts of the system, bad architecture and so on. It had to be improved and implemented in the next sprint to meet the requirements.

4.6 Why this project?

To find a relevant case is not always easy. Therefore, I had to put a lot of effort into finding a project or case that was relevant for my questions. First, I contacted another company with another project that I was supposed to research, but because of

restructuring in the project they used too much time before I got a definite answer on whether I could study that case or not. Therefore, I had to look around for another project.

I used some of my contacts from a company I have previously worked for. They had several options for projects I could study. After an evaluation of what I needed from a project, they presented me with a couple of cases that could be relevant for my study.

The choice fell on this particular project because they had met some interesting challenges that seemed like a good choice for further investigation. The reason that this project was chosen was because of its obvious relevance. The project was started with no or few requirements. The requirements had to be found in cooperation with the customer in a series of meetings. Unlike “regular” scrum projects where you have as little documentation as possible, it was in this case chosen to have an analysis and design project to find the requirements and develop an understanding of what the customer wanted from the system. This is also a unique case because of the change in framework and approach from one part of the system to another. This makes it very interesting to study.

5 Results from study

There were found several interesting results from the study. Most of them from the interviews I held, but also some interesting observations from the documents and meetings I was able to observe and review.

5.1 Use of agile methodology and architecture framework

The architecture framework IAF was applied to the project and used to some degree. For development, the agile development methodology Scrum was chosen. The level of satisfaction with these choices can be viewed in Table 4.

Table 4: Use of methodology and framework

Topic	Customer	Functional architect OFS	Functional architect DFS	Functional responsible	Scrum master	Project manager
Satisfaction with level of agility	Satisfied with the way they use agile now	Satisfied	Satisfied	It works well with the constraints that are set	Satisfied	Satisfied
Satisfaction with the use of IAF	Satisfied	Satisfied	Worked well	Satisfied	Nice way to work	Satisfied

5.1.1 Agility of project

The project used a many of the agile principles and the methodology Scrum. Some changes was nevertheless made to make the methodology match the project and the contract that was used. The level of agility applied in the project is, based on the satisfaction level of the interviewees a suitable level that worked well in this project.

The functional architect OFS summarizes the use of agile in the project in a good way

> *"I think it is agile in the right level. It is an effective method to avoid small things. We are doing this for a business that wants to control the project. And pure agile projects has a way of drifting of in a direction that you are not aware of because it is user driven".*

How agile the project is depends on how agile is defined. According to functional architect OFS:

"Some define true agile as formulating a problem and putting aside a given number of resources in a period of time and do as much as possible during the given time. This is not exactly how we work. We want to know what the project consists of on a certain level. There is not room for everything in the project; one must have a certain level of control of what one is doing. Especially when it comes to the customer".

Project manager said that:

"Totally agile is probably more suited for projects in internal organizations where they are adding more features to something that is already in production. In this project it is not possible because there can be no errors when the system is in production and there are strong customer requirements in relation to project cost, milestones (time) and scope".

These are some of the reasons why completely agile is not used for the project.

5.1.2 IAF: choice of architecture framework and affection on requirements

The architecture framework IAF was introduced for iteration 2 based on the experiences from iteration 1. The interviewees had an overall high satisfaction with the use of the IAF framework. Most of them uttered that they did not have an in depth knowledge of what IAF meant for the project other than the use of IT-services and work tasks. However, the satisfaction with the use of IT-services and work tasks was very high on both the customer and vendor side. It was pointed out to me that they were very happy with the use of IAF, but it was mentioned that other architecture frameworks might have worked just as well as long as they settled on a mutual language for communication.

The high satisfaction with the use of IAF might be explained by IAF according to the functional architect OFS, only was used to some degree. They chose to use the parts of the framework they regarded as suitable in this project. This included IT-services, work tasks and case flow, which was a big part of the functional representation of the system that was to be developed.

5.1.2.1 *Why IAF?*

The interviewees gave several reasons for why the framework IAF was chosen for the project. It was because IAF facilitated a:

- Mutual language for communication
- Guidelines to communicate wishes and needs from business to development in a good way
- God method for estimating complexity of IT-services

- Appropriate for every project as long as it is introduced early in the process
- Open for changing requirements

According to functional responsible they used IAF to get a mutual jargon they could all understand. They used it as a guideline to communicate the wishes and needs from business over to development in a good way. It could have been something completely different that would have worked just as well according to functional responsible. However, functional responsible pointed out that what really worked with IAF is that you get a mutual language. In addition, it was a good way to estimate the complexity of the IT-services.

The functional architect OFS thought that IAF was appropriate for the project:

> *"My opinion is that it is suitable for every project really, but it depends on where we arrive in the project, if someone else has made a long list of user stories it is limited how much we can use it. However, to try to take the user stories and make some components from them. For example, decide that three user stories are one screenshot. To do that job is important"*

The scrum master said that he was pleased with the use of framework but did not have in-depth knowledge about the difference between this and other frameworks. However, the scrum master thought that IAF opened for changing requirements.

5.1.2.2 *Challenges that were met*

The introduction of IAF was according to the project manager a bit challenging in the beginning due to introducing a new framework and jargon for communication. However, after an adaptation period where they were able to readjust to the new framework it seemed like it was a suitable choice.

There were also pointed out some negative aspects with the use of IAF in the project.

- Test people complained a little because they had to work a little different
- Challenges with big DFS because they were made on IT-services and work tasks, which require a lot of detail. Resulted in documents that were up to 100 pages
- Customer thought the IT-services had somewhat difficult names

According to the project manager the test people complained a little due to the use of IAF. This was because they were forced to work a little different from what the test system was rigged for. They had more to do because they could not build tests based on user stories anymore. If they had used user stories they could have tested each user story, but because user stories were put more aside they had to test against a whole chapter from the DFS and every detail in the chapter instead. However, increasing

number of test cases was somewhat natural due to the increased complexity and size from the first iteration.

The functional architect of the DFS also saw some challenges because the DFSs were very big. Functional architect DFS said that “I think the biggest DFS we made was around 100 pages”. They got big because they were based on IT-services and work tasks. These required a lot of detail. This results in a big document for the customer to read and approve.

The customer thought that the IT-services had somewhat difficult names, but found it possible to navigate through. They often had to navigate into the OFS to find out what the IT-service was supposed to do because it was not intuitive from the name.

5.2 Requirements process

The whole process of specifying the requirements and looking into different subjects when it comes to requirements specification is very interesting when wanting to answer RQ1: How thorough should the requirements specification be before development starts? In addition, RQ2: How can one make the specification thorough enough?

In Table 5, it can be viewed what the different interviewees thought about how the requirements were presented in the project, the changes that were made and the processes that surrounded the specification of requirements.

Table 5: Satisfaction with requirement process

Topic	Customer	Functional architect OFS	Functional architect DFS	Functional responsible	Scrum master	Project manager
Satisfaction with representation of requirements	Satisfied	Satisfied	Satisfied	Both yes and no	Satisfied	Very satisfied
Satisfaction with change management on requirement	Semi satisfied	Very satisfied	Satisfied	Satisfied	Satisfied	Satisfied
Satisfaction with changes from iteration 1 to iteration 2	More satisfied	Satisfied, easier to understand	Better than before	Better than before.	There has been no changes	Satisfied. Better than before
Level of detail in the requirements specification when using agile	Project and organization dependent	Project dependent	Should be high	Should be 100% when development starts	Should be high	Should be high

5.2.1 Representation of requirements in project

The thoughts on how the requirements were expressed when the interviews where held, which is during iteration 2 can be viewed in Table 6. From the table one can see that the requirements were at the moment of the interviews presented in a level of detail that was perceived more or less as sufficient by all of the interviewees**(6/6).** The requirements were also handled in such a way that most of the interviewees were very pleased. The developers uttered some dissatisfaction with the level of detail in the DFS because they sometimes had to use more time than wanted on interpreting some of the functionalities expressed in the DFS, but it was pointed out that it usually contained a satisfactory level of detail.

Table 6: Thoughts on representation of requirements

Topic	Thoughts
Customer	Is very pleased with how the requirements are presented at the moment
Functional architect OFS	It is ok, but could just as well have done it a different way
Functional architect DFS	Very good to have screenshots in the specification. Good specification, but not perfect.
Functional responsible	Found a middle way that is suitable for everyone
Scrum master	Usually a satisfactory level of detail
Project manager	Good method that is easy to understand

5.2.1.1 *Thoroughness of requirement specification in project*

The requirements specification was perceived as very thorough by most of the subjects that were interviewed, but with some minor dissatisfaction from some of the subjects. By thorough is meant that the level of detail of the specification is very high. In Table 7 a summary of the interviewees' thoughts on how thorough they managed to make the specification can be seen.

Table 7: Thoughts on thoroughness

Topic	Thoroughness
Customer	Worked well with the level of detail, but might be too thorough
Functional architect OFS	Very thorough when the DFS is done
Functional architect DFS	As thorough as it can be
Functional responsible	Usually good enough, but sometimes different scenarios are forgotten
Scrum master	Pleased with the level of detail, but would like it to be more thorough if possible
Project manager	Very thorough and great level of detail

The customer had some objections when it came to the thoroughness of the specification, but were very pleased with the level of detail they manage to obtain. The customers objection was that sometimes there were too many details in the specification. Too many details resulted in the developers not being able to use their imagination when developing. The customer said that:

> *"We miss a more critical view from the system developers. These young innovative people straight from the university. This might be what is sacrificed by creating a production factory like we have done here".*

The functional architect for the OFS thought that the requirements specification was very thorough when a certain point in the process was reached. This was when they

reach the DFS. This was where the details were added. Before this point, when they were only expressed in the OFS, they were not yet very detailed.

The functional architect of the DFS stated "I think the requirements specification is almost as thorough as it can be". Functional architect DFS further pointed out that the only thing that might be missing was a couple of more process drawings in the DFS.

The functional responsible stated that requirements were usually good enough, but not always. In the DFS phase, one found that the user stories in itself were not good enough. They were on such a high level that it was impossible to develop on them.

> *"But in the DFS and IT -services it is usually detailed on a good enough level, we sometimes see that we have forgotten to think about different scenarios. This is often caught by the developers which come back with questions on how they should handle the different scenarios". (Functional responsible)*

These are cases where they missed something in the DFS and they had to have a quick coordination with the customer. The coordination was usually done in a minute. These type of events could possibly have been avoided if the DFS was more thorough. But making it even more thorough could have led to a planning paradox. By planning paradox one means that you plan for every possible outcome, but use too much time. Planning for everything might take forever. Therefore, the functional responsible said that

> *"Instead we are quite sure that this and this will work and we go for it. So it is a calculated risk".*

The functional responsible also pointed out that

> *"I don't think it is possible to be a 100% sure. You will always have questions and secondly it would have been very expensive".*

The scrum master was also pleased with the DFS. He said that it was very thorough. But would have wanted it to be a little bit more detailed. Scrum master would have wanted the DFS to cover more error situations and not only "happy paths". The scrum master also pointed out that one has to think about how much time is available when making the specification. In a perfect world one would cover every possibility, but it might not be necessary or smart to spend that much time on making the specification. One has to stop at one point.

The project manager said that the specification was very thorough. This was because they had the opportunity to write the specification very elaborate and very detailed, more so than in many other projects. The level of detail was raised from iteration 1 to iteration 2, by adding IAF. Therefore, the project manager was very pleased.

5.2.1.2 *Suggested improvements to the specification*

There were suggested some improvements to the specification that would probably have made it a little better. These were:

- Cover not only "happy path"
- Process drawings in DFS before development

Scrum master would have liked the DFS to cover not only "happy path" or ideal case. He wished that it would also cover the case of an error situation. This is something that is seldom thought of in advance, but the developer has to deal with when the system is developed and discovers that an error situation is not specified. It is not possible to specify them all, but the ones that happen most often should be specified before development starts and in the DFS.

A possible further improvement to the DFS that was proposed by the functional architect DFS was process drawings. The scrum master agreed that this might be an improvement because they help with the visualization of the flow. They also help get a better overview of what possibilities there were and what type of challenges one might encounter. Functional architect DFS said this about process drawings:

> *"We are good at making them afterwards, after development or at the same time, where developers make their own process drawings".*

Functional architect DFS further said: «I would like to have them before I start specifying so that I can see dependencies». This is because the system was getting very complex. It was hard to see where one thing would affect a different part of the system. Process drawings would probably have helped see this. Therefore, it was something that was planned implemented for iteration 3.

5.2.1.3 *The significance of requirements specification in the project*

The methods that were used for specifying the requirements in this project were viewed as very significant for the project success. This includes the meetings where they walked several rounds with the customer to find the best solution for their problems and the use of an OFS and DFS with the framework IAF.

The specification was important for the project success because:

- Obviously necessary to know what one was supposed to develop
- The OFS helped set the scope and find potential dependencies early
- The DFS was one of the main reasons for their success:
    - Made it possible to catch uncertainties or potential errors at an early point
    - Delayed the final (non-cost influencing) details as long as possible

- Screenshots in the DFS helped a lot for the visualization of how the finished product should look like
- Developers did not have to ask many questions due to detailed description of screenshots and the functionality they provided

As the scrum master stated:

> *"Without the requirements specification there would have been no project, it is necessary to know what one is supposed to develop".*

Scrum master continued to state that the DFS was one of the main reasons they managed to deliver what they delivered on the time they had available. The DFS made it possible to catch questions and uncertainties at an early point in the process. If the questions were not asked before they started development, it would have increased the cost. This is because developers, according to the scrum master, should not do anything else than follow the specification when they develop. If the developers were left with the questions, due to the questions not being asked at an earlier point, there would have been loads of wasted time. If the errors in the requirements specification managed to reach test, it would have cost even more because you would have had to go back to the developers, and maybe all the way back to the specification and discuss again. It might even be necessary to make the functionality again and test it all over again.

> *"I think the process we have is very good in the sense that we make the specification a sprint before and it is as detailed as it is. Therefore, we catch most of the possible mistakes there. I think that is very important" (Scrum master).*

The functional architect stated that

> *"The specification itself is very important. It is what everyone works around. One way or another we needed a requirement specification. It is important to make sure everyone is as close as possible when it comes to what everyone images the project to be. It helps very much that there are screenshots in the specification that indicates how it should look like".*

From this, we can understand that, the written specifications is not the only thing that is important, but a visual representation before the development starts is also important. A lot of the honor of the project success was given to the DFS and the DFS being as detailed as it was.

5.2.1.4 *Representation of requirements in documentation*

The OFS that was introduced for iteration 2. My impression from reading the OFS is that it is very easy to read, it represented a nice overall image of the functionalities and

an overall image of integration and information model. In addition, it defined which screenshots should be included, but did not go into detail on how they looked like or what they should do .This made it easier to see the bigger picture when it came to how the user stories interact. It made it easier to connect them and see how the whole system was put together.

The DFS, which was the final description, seems very important and was viewed as important by the interview subjects. The DFS goes into great detail and expressed exactly how the different functionalities of the system should be. It supplemented the written description with a detailed screenshot and went into detail of what the screenshot should do. This is from my point of view easy to understand and according to the scrum master; it was something that helped them when they were developing the system. This was also something that helped the customer see and understand the requirements in a way that was understandable for them.

A detailed description like this seems like some of the reason for the great success of the system and removed pressure from the developers during the sprint; due to them not needing to focus on interpreting the requirement as much as they might would have needed if the documentation was not as detailed. This is also the impression I have gotten from the interviewees.

5.2.2 Changing requirements in project

According to functional responsible there were not many changes to the requirements in this project. It was more detailing and refinement of the requirements in the DFS phase. This was because in the DFS phase, they got the understanding of the process. It was also a forum for new thoughts to emerge. In this particular phase, there were loads of small changes, but the overall requirements stayed more or less the same. According to the functional responsible the fact that there were few change orders might be because of the DFS phase and the DFS. This was because they got an actual relationship with the functionality that was being developed by revisiting it repeatedly. However, there were some changes to the requirements in the project. Most of the changes happened in the DFS phase or before. This was exactly as intended. Changes happened here because:

- Requirements were detailed a long time ago with few details
- The customer no longer wanted what they wanted at the time the requirements were detailed

The requirements were detailed a long time ago and had few details. The project manager said that:

> *"For example now that we have worked on a complete review of the product backlog, we see that many of the elements that were detailed a year ago, is outdated now. They expire".*

Another reason for the changing requirements was that the customer did not always want the same as they wanted in the beginning. Sometimes they wanted to remove some functionality completely and sometimes they wanted to add some functionality.

> *"Sometimes we don't understand why we wanted a feature to begin with. It was wanted at the beginning, but when we revisit the requirement, it sometimes seems completely absurd. So it is good that it is possible to change our wishes" (Customer)*

Most of the potential changes were because of the high level of detail and close customer interaction, caught before they started the sprint. However, there sometimes happened changes in other phases of the project. This was due to:

- The DFS lacked some information the developers needed or there were logical flaws in the specification
- Customers changed their mind after functionalities were implemented

Even though the requirements were supposed to be locked for a sprint when the sprint started. There were sometimes according to the scrum master, changes to the requirements during the sprint. The making of the DFS was one sprint ahead of the development team. Sometimes there were details missing or the DFS was not detailed enough so that the developers had to interpret the specification that was made. When this happened, there sometimes occurred minor changes. For example minor changes in the text. Sometimes there were logical flaws that were discovered that forced the developers to walk through the functionality with the customer again to find out how it could be done in a satisfying manner. This resulted in a change in the specification when they arrived at an agreement.

5.2.2.1 *Satisfaction with how requirements were changed*

There were some dissatisfaction with the way changes were handle in the project. The customer had some dissatisfaction with all the technicalities when they want to change something, but also understood that the technicalities might be needed. The satisfaction with how requirement changes were handled can be viewed in Table 8.

Table 8: Satisfaction with changes in requirements

Topic	Thoughts
Customer	A lot of technicalities when introducing changes
Functional architect OFS	Very organized
Functional architect DFS	Sometimes not enough time
Functional responsible	Works well
Scrum master	A little frustrated due to many last minute changes
Project manager	Works well

The customer thought there were many technicalities when introducing changes because of a rigid change management regime that was time consuming. However, the customer was very pleased that they were able to do changes directly on IT-services and not user stories. This was because the user stories were too detailed. The customer also pointed out that it became easier when using IAF and IT-services than having to browse through a huge amount of user stories. The customer also found that it was easy to introduce changes as long as it was enough time in the project. There were a couple of challenges when it came to organizing changes and getting them efficiently into the project, but introducing changes was not difficult it was harder to prioritize out something else.

According to the functional architect DFS, there were more changes to the requirements than the developers would like it to be. There were a couple of complaints because there was a sudden start on iteration 2. This was because they expected that they would start with one functionality that was almost completely specified, but then there was a change in the last minute. This lead to challenges when it came to time. The scrum master also mentioned that there was a little frustration due to many last minute changes.

5.2.3 Changes from iteration 1 to iteration 2

From iteration 1 to iteration 2 of the project, there were introduced some changes. The level of satisfaction with the changes can be viewed in Table 9.

Table 9: Satisfaction with changes from iteration 1 to iteration 2

Topic	Satisfaction with changes
Customer	More satisfied
Functional architect OFS	Satisfied, easier to understand
Functional architect DFS	Better than before
Functional responsible	Better than before.
Scrum master	There has been no changes
Project manager	Satisfied. Better than before

The customer said that the change into IT-services and the use of IAF was very pleasant on a higher level. It was pleasant that they no longer needed to keep track of the user stories. In addition, the customer was very pleased that they could do changes on IT-

services instead of user stories. The customer pointed out that everything was easier in iteration 1 because the project was not as complex yet, but because of the growing complexity of the system, it worked better with IT-services than user stories.

According to the functional architect DFS the change was not that big. Work tasks and IT-services was used in iteration 1 to describe the user stories more thorough. It was not that big of a change, but according to the functional architect of the DFS if they had used user stories the way user stories are used in other projects it would have been very negative.

The framework IAF was introduced an initiated by the architects. The introduction of IAF resulted in a couple of changes within the project due to them having to start working a little differently, but the project manager said that it has been well received.

5.2.3.1 *Why change?*

The changes was initiated because:

- Needed to be clearer on a higher level
- Make something the customer could understand
- Make something the developers could understand
- Impossible to maintain all the user stories
- Not covering enough with user stories

In iteration 1, they changed from a list of user stories to a more structured representation. This was according to the functional architect OFS because they needed to be clearer on a higher level. They needed to be clearer on what they were going to deliver. A user story only tells you that there should be possible to do something. However, with the screenshots you define something the developer can look at and make.

> *"Then you have managed something that is not only a requirement for something that should happen, but also something the customer can understand. That is one of the reasons" (Functional architect OFS).*

They found that this was the smartest way to do it early in iteration 1. The user stories further described how things should be connected.

According to the customer, they had some impact on the change to IAF and IT-services. When they were supposed to approve iteration 2, they saw that to go through all the user stories and be sure that they covered everything was a very extensive work. "

> *"We had for example 300 user stories and we had to see what was missing. That is essentially, what you do when you go through it. It is*

always something missing, but it is hard to keep track of everything" (customer).

According to functional architect of DFS, they changed from user stories to IT-services and IAF because there were so many user stories that it was almost impossible to maintain every one of them. It became impossible for the customer to read and make sure that everything was included in the specification. It became too complicated for using only user stories. After the change, it was easier for the customer to see a bigger picture of what they would actually get from the system.

5.2.4 Specification when using agile

Most of the interviewees agreed that it is important to make a detailed requirement specification even though an agile development methodology is used. Some of the interviewees have not worked on other projects before, but most of the interviewees thought that the level of detail they put into the specification in this project was a satisfying one.

Most agreed that the specification should be very detailed when development starts. Almost to the point where the developer can read the specification and develop with no further questions. This is because questions at this point in the process will cost more time and money. Before this point, it does not necessarily have to be very detailed. It was also pointed out that the thoroughness of the specification depends on both type of project and organization.

In Table 10 the different thoughts of the subjects are presented.

Table 10: Views on how thorough requirements specification should be when using agile

Topic	Thoughts
Customer	Depends on the organization. If organization is saturated on change, there should be great detail before development starts
Functional architect OFS	Not necessarily important. Depends on the project
Functional architect DFS	Important even when using agile. Exceptionally important in this project
Functional responsible	Can be loosely defined right up until development starts. Then it should be 100%.
Scrum master	Important to make it thorough even though it is agile. More important in bigger projects
Project manager	Make it thorough right away

If the organization is one that is used to doing projects, the customer thought that they could keep the specification on an overall level. If the organization on the other hand is one that is saturated on change, is reluctant when it comes to changes, or there is a

project that requires big changes, it would probably be beneficial to go into great detail before one starts developing. The functional architect of the OFS said that it is:

> *”Not necessarily more important to make a detailed requirement specification than using user stories when developing using agile. It depends on the project”.*

The functional architect OFS further stated that:

> *“We do not want all the details, we find them afterwards. We don’t need all the details to plan the scope of the system”.*

By this, he meant that all of the details are not needed right in the beginning, but can be found along the course of the project. It was also pointed out that:

> *”I think that right before it is being developed it must be 100% detailed, but until then (until it is inside a sprint), it can be as loosely defined as you want it to be” (Functional responsible).*

Even though the functional responsible thought that it does not have to be a 100% until right before development he acknowledged that this might lead to some difficulties.

> *”The vaguer it is specified, the later one discovers possible dependencies and occurrences that will slow down or even stop the development. For example if you detail a user story that is supposed to be developed the coming sprint and you see that software that takes 10 weeks to order is needed to complete the functionality, this will slow down or maybe even stop development” (Functional responsible).*

This is one of the downsides of waiting too long with detailing the requirements. One alternative he suggested was to detail more than needed so that you had other elements that could be developed. Like a buffer of elements you can pick from if some of the functionalities that were meant for the upcoming sprint were not possible to include.

The scrum master said that “It is important to make a thorough specification even though the project is agile”. The scrum master further said that:

> *“Previously I have worked in a firm where they were a loot “looser” when it came to the requirements specification. I usually wrote the requirement specification myself in cooperation with the customer and developed it myself afterwards. This might work for small projects, but not for bigger projects”.*

One of the other subjects based their experience from this project, due to no previous experiences with projects.

"I only have experience from this project, but I will say that it is very important to make a detailed description even when using agile" (Functional architect DFS).

Even when using agile methodologies the project manager thought that it is important to make the specification thorough right away. Project manager stated that

"My experience is that it is worth the investment to make the requirements specification thorough right away"

This is to avoid complications that might arise if they wait too long. According to the project manager, you will see that a number of things happen, if the requirements are not specified thoroughly enough before development starts:

- Number of mistakes increase
- Time used will increase
- Quality will be lower
- Costs will be higher
- Discover dependencies late
- More wasted time by having to clarify misunderstandings at a later point
- Potential conflicts with customer due to misunderstandings due to specification that is not thorough enough.

The project manager thought that it should be done both before the project is planned, to find the scope of the project and before the sprint starts. Even though this requires the use of more time at the beginning of the project. The project manager pointed out that it is an investment you will benefit from in the long run.

5.3 User stories

To answer research question RQ3: *When are user stories sufficient and when are other techniques or tools required?* I have used the project as an example and drawn conclusions from their experiences with the use of user stories in this project.

5.3.1 From interviews

From the interviewees I got a lot of relevant information. They said that in the beginning, the user stories were used to set up the contract with the customer, but the functional architect saw that this was not sufficient:

- The project was very big which resulted in an unmanageable amount of user stories
- Customer experienced difficulties when wanting to do changes on requirements
- User stories were not covering enough
- User stories were too user oriented

- The user stories where not connected in a good way and did not cover the different dimensions
- The user stories did not get read after the DFS was made

The reason why user stories were not sufficient was amongst other the size and complexity of the project. They ended up with an unmanageable list of user stories that was hard for the customer to navigate through and make changes on. Change management on the user stories was difficult. This was due to small details like for example that the color of a button could change many times, this again resulted in the need to change many user stories. Therefore, it was decided that these details were left out until right before the sprint, where they were specified in the DFS.

It was also seen that user stories were only fragments of what was wanted. This was discovered when they first took the user stories from the context they were in at the beginning of the project. They started with the user stories for the contract and saw that they needed to limit it in some way. They were taken out from their context and it was seen that further use of only user stories would have resulted in many discussions.

> *"The user stories weren't covering enough. Because they could be interpreted one way, they were detailed enough, but it didn't cover the different dimensions on an overall level" (Functional architect OFS).*

It was also hard for the vendor to see connections and the scope of the system.

The user stories became a little bit too user oriented and resulted in the user specifying what they wanted from the way they did things at the moment, not optimizing routines and processes.

This resulted in:

- Introduction of IAF, IT-services and work tasks
- User stories being made from screenshots not the other way around
- User stories being used only as a guide through the DFS document
- IT-services more important than user stories
- DFS worked as user stories and was what the developers worked from in the sprint
- Developers did not use user stories in the sprint

It was decided to express the system through a more functional oriented method and applied the architecture framework IAF with the use of IT-services and work tasks. The user stories were expressed through the IT-services, but the IT-services were weighted higher than the user stories. Instead of working on user stories when the final specification was being made (during the meetings and during approval of the final DFS) they approved the IT-services and textual description of the service instead.

Functional architect DFS said that

> *"In iteration 2 we had some user stories, but we ended up throwing them away and writing the user stories from the screenshots we made".*

In the DFS, the user stories worked only as a guide and the IT-services contained the final description of what the system should contain. Functional architect DFS further said:

> *"My impression is that the user stories often doesn't get read, this is not the most important part. The rest of the document is".*

Functional architect DFS further said this about the user stories in the DFS:

> *"They are kind of redundant, but we had to use the user stories that were already specified for something" (Functional architect DFS).*

The change was very welcomed by the customer. It became according to the customer more flexible to do change orders, when IT-services were used. If change orders all the way down to user stories had to be done, the customer felt like change orders would have taken up all their time due to all the small changes. The user stories were somehow too detailed.

The developers had no relationships with the user stories. When they developed, they only looked at the DFS to get the final detailed description of the system. The scrum master said that he knew user stories existed in a document somewhere, but they only looked at the IT-services, work tasks and the DFS.

5.3.1.1 *When to use user stories*

User stories is a concrete way of describing system requirements. This makes user stories a good thing, but not necessarily suitable in this particular project. User stories can according to functional architect OFS be used when one wants to describe concrete system requirements as a way to map and document pure user requirement. Functional architect OFS said that he was not quite sure when one should not use user stories, but that sometimes it is not a complete enough description of the system. User stories make the description a little fragmented, but it is still useful. Functional architect further said that to think that user stories are everything you need, might not be a good thing.

5.3.2 Observations around the use of user stories

During the meetings I was able to observe, I found that the discussions were not on user story level. They discussed the screenshots and the different functionalities the screenshot was to contain. If functionality was supposed to be changed, they discussed the IT-service at the meetings. This supports the findings from the interviews where

interviewees view the DFS and the description in the DFS as most important and the user stories as somewhat redundant.

5.3.3 From documents

In the DFS the user stories were at the beginning of the document. They reference the IT-service that covered the user story. Not all of the user stories were included under an IT-service. When this happened, it meant that the user story was not relevant any more. This also supports the findings from the interviews and observations.

5.4 Non-functional requirements

There has been no observations or viewing of documentation regarding non-functional requirements. The results are therefore based on interviews alone. As mentioned, non-functional requirements are something that is poorly specified in agile methodologies. Therefore, it was particularly interesting to see if the subjects thought these were handled in a good way in the project.

From Table 11 it can be seen that there is some dissatisfaction with how the functional requirements were handled in the project. This is something that is very common. The non-functional requirements are often neglected and the focus is on the functional requirements because the functional requirements deliver visual results.

Table 11: Satisfaction with handling of non-functional requirements

Topic	Non-functional requirements handled in a good way?
Customer	Could have been handled in a better process
Functional architect OFS	Could have been designed in at an earlier point in the process
Functional architect DFS	Thinks they are taken into consideration in the sprints
Functional responsible	Not sure how this is done
Scrum master	Should have had more time. Might come as a surprise in the end
Project manager	It is handled

The functional requirements were worked with on a daily basis and the non-functional requirements might be thought of at the end of the sprint and "come at the end as a surprise" as the scrum master said. The reason for this might be that there were far more functional requirements than non-functional. In addition, the functional requirements were easier to see results from.

5.4.1 Suggested improvements

From the interview there was suggested some improvements to how they handled the non-functional requirements in the project. The suggestions were:

- Set aside more time
- A meeting before every other sprint to discuss the non-functional requirements
- Meeting before each sprint to find out if something that is supposed to be done during the sprint does not support the functional requirements
- Design the non-functional requirements into the system at an earlier point

The scrum master supported the thought of better routines and setting aside more time on the non-functional requirements. He suggested a quarter or half an hour each sprint where the technical responsible could find out if something that was supposed to be done in the sprint did not support the non-functional requirements that were defined.

The customer suggested there should be a meeting not before each, but before every other sprint. In this meeting, they could go through the functionalities that were supposed to be implemented during the next two sprints, how it affected the non-functional requirements and how the requirements that were affected should be handled. This would according to the customer probably improve the handling of the non-functional requirements.

The functional architect OFS thought that if the non-functional requirements were designed into the system at an earlier point in the process, this would probably have helped avoid the last minute handling of the non-functional requirements. However, to design in the non-functional requirements at an earlier point requires a huge amount of knowledge if using a lot of time should be avoided. Functional architect OFS pointed out that it all depends on routines. In addition, you have to decide on the non-functional requirements like for example performance at an earlier point. When this is decided, one should also be more specific on what the developer must do to maintain the non-functional requirements in a satisfying way. This was not necessarily done at the point of the interviews.

5.5 Overall satisfaction with project

It is important to point out how satisfied the different interviewees were with the project. This is because their satisfaction with the project is closely linked with the success of the methods that have been used. In Table 12, the satisfaction level of the different interviewees are presented.

Table 12: Overall satisfaction with project

Topic	Satisfaction with project
Customer	Over all expectations
Functional architect OFS	Satisfied
Functional architect DFS	Very satisfied
Functional responsible	Worked well
Scrum master	Satisfied
Project manager	Works well

As can be seen from the table, that the customer was very pleased with how project was going at the point of the interview. The different subjects from the vendor were also pleased with how things were going.

6 Discussion

The conclusions that can be drawn from the results are discussed in this chapter with respect for each research questions.

6.1 Question 1

The main question is:

RQ1: How thorough should the requirements specification be before development starts when using agile?

How thorough the requirements should be, depends on the project. In a big, complex project it is important that the requirements are well understood so that the customer get what they want. It is also important that there is some flexibility. To have the entire requirements specification completely locked long before development is started is not a good method. It is important that there is some agility when it comes to changing requirements and in the processes that lead up to development. This has been proven to work in this particular case.

The method that has been chose in this project seems like it has worked very well. The interviewees were very pleased with how the project was going at the time of the interviews. The reason for this is amongst other due to high level of customer interaction and the agility when finding the requirements. It is also due to their chosen method for detailing the system, where they detail 80% of the requirements before the project started, and then the remaining 20% right before the development sprint. In respectively the OFS and DFS.

In for example a project where close customer interaction is not possible, the chosen method will probably not work that well. This is due to the finalization of requirements that used a just in time principle, where the requirements were not completely finished until right before the sprint started. If the customer is not available, one should go for another approach.

6.1.1 Not totally agile

First, I want to point out that the project development was not completely agile in this particular case. It was chosen to use the agile development methodology scrum, but only the elements that were viewed as appropriate for this project was included.

The subjects were very happy with the degree of agile that was used. They could not have been more agile than they were according to some of the interviewees. Others meant that they could possibly have opened up to not having as many details before they started development so that the developers could have had some more artistic freedom during the sprint. The method they chose, where all of the details were

finalized before the development sprint started, put some constraints on the developers. Where they had to do as they were told by the DFS with preferably no questions asked. This made the sprint in itself somewhat waterfall like.

In the background section, it was pointed out some challenges that has been met with the requirements specification when using agile development. The challenges and the impact they had on the project are pointed out here:

1. Problems with cost and schedule estimation: The problem with cost and schedule estimation was avoided by specifying 80% of the functionalities before the project started. These 80% were the 80% that affected the cost of the system.
2. Inadequate or inappropriate architecture: The use of IAF and OFS helps avoid this, and there were no problems indicated concerning the architecture, by the interviewees.
3. Neglect of non-functional requirements: It was pointed out that the non-functional requirements were neglected to some degree also in this project.
4. Customer access and participation: In this project, the customer was very available. This was pointed out by several of the interviewees. Therefore, customer access and participation was not a problem. However, this is something that is outside the vendors control.
5. Prioritization on a single dimension: Because they mapped the architecture before they started development, they added the opportunity to scale and evolve.
6. Inadequate requirements verification: In the project it was a focus on verification of the requirements through revisiting them several times before the functionality was implemented into the project.
7. Minimal documentation: It has been a great focus on documentation in this project. Which was necessary due to the complexity of the system that was developed and because of what was decided in the contract. One might say there was too much of it due to DFSs that became up to 100 pages long.

Even though the project was not entirely agile, the methods that were used might be applicable to other big semi agile projects that require many details in the specification.

6.1.2 Factors to consider when deciding how thorough specification should be

The interviewees pointed out some factors that the detail level of the specification should depend on and therefore, be considered before deciding how thorough the requirements specification should be:

- Project:
    - Size
    - Complexity
    - Available resources
    - Importance of system

- Organization and customer
    - Is the customer available?
    - Is the project in-house or for external customer?
    - How mature is the organization?
- Type of contract being used

If the project is big in *size*, it is often also a *complex* project. These factors often results in the need of finding the dependencies between for example different functionalities in the system and dependencies when it comes to hardware. These dependencies might affect in which order one can start developing. Some functionalities in the system might also depend on for example hardware that can take several weeks to get. It is important to catch these type of dependencies early to avoid bottlenecks when developing. A bottleneck in this case is when one dependency stops the whole development and prevents the available resources to work as efficient as possible. One of the benefits of agile is that you should be able to change the order of functionality that is implemented if necessary, but some dependencies might create bottlenecks even when using agile. Therefore, it is important to have these mapped at an early point.

Another important factor is how important the system is. Questions that can be asked with regards to importance is: Is it important that the system does not fail? How important is this? If it is critical that the system does not fail, it is important that the requirements are thoroughly described to find potential flaws before one starts developing.

When considering how thorough the specification should be it is also important to consider what kind of organization and customer you are dealing with. When there is a customer involved, it was pointed out by the one of the interviewees that it is important with a thorough specification to avoid misconceptions, which will result in an unhappy customer.

It is also important to evaluate the availability of the customer. If the customer is not available for reviewing and further detailing of requirements, more of the details need to be set in stone at an early point. Alternatively, if the project is for example an in-house project it is easier to ask questions and find the requirements at a later stage.

The maturity of the organization the system is developed for is also a factor one has to evaluate according to one of the interviewees. If the organization is mature it is thought that one can keep the specification on an overall level, but if the organization is saturated on change or has to preform big changes, the requirements probably need to be more detailed at an earlier point.

The type of contract that is used also put some constraints on how much documentation and specification has to be done up front. Type of contract depends on customer, project and vendor.

6.1.3 Before development starts

There seems to be a difference of opinions when it comes to how thorough the specification should be up until the point where development starts.

The functional responsible said that right up until development starts the specification does not have to be thorough at all. It can be as loosely defined, as you want. The specification does not have to be a 100% before right up until development starts. He pointed out that there are of course negative aspects when using this approach. The result might be that dependencies might show up too late.

The other subjects seemed to agree on that the earlier you specify the requirements, the earlier you find dependencies and possible problems, but how carefully you have to go into the specification before starting development they did not agree on.

In this project the interviewees have also found that it worked well to start writing down the functional specification of the system and decide on the function requirements on an overall level at an early point. This represented an 80% description of the system functionalities and helps the customer, and the vendor see the overall picture, find the scope and price at an early point. The last 20% was postponed until the DFS. The interviewees thought that the use of the DFS and postponing the last 20% of the details until right before the sprint started, helped reduce the amount of changed orders and is some of the reason why the project has been successful up until the point of the interviews.

6.1.4 When development starts

There seems to be a somewhat agreement that when the sprint in itself starts, the description should be at such a level of detail that the developers could use their time on development and nothing more. Their time should not be used on interpreting the requirements during the sprint. If the developer has to use time on interpreting requirements specification, they might need to go all the way back to the customer, which is a big waste of time and money. However, not having too many details so that the young and innovative developers could find, possibly better solutions was wished by the customer.

Most of the interviewees agree that even though one uses agile development methodologies, the specification should be very thorough.

6.2 Question 2

Question 2 is based on experiences from the case I have been studying.

RQ2: How can one make the specification thorough enough?

6.2.1 How to make it thorough enough: experiences from project

To make the requirements specification thorough enough for a big and complex agile project one can draw some experiences from the case that was studied. These are:

- Good processes to get the requirements from the customer
    - Series of meetings before final decisions are made
- Good cooperation with the customer: important with active and involved customer
- Start with a high level description that involves elements that affect costs and make more detailed description at a later point to avoid a many changes to the specification
    - 80% description of functionalities before project start
    - Last 20% right before sprint start
- Describe the work tasks, IT-services and include screenshots with detailed description

The meetings that were used to draw the requirements from the customers and the processes that were used here has proven very successful for this project. This is something both the vendor and the customer has been very pleased with and the general opinion is that being as agile as they have been when it comes to finding and specifying the requirements was some of the reason for the projects success.

Another key factor that helped them make the description thorough enough was that the customer was very available during these processes and very involved. The customer participated in regular meetings and was open for further questions when needed. The general opinion from the vendor was that they could get in touch with the customer whenever they needed.

It was also pointed out that making the OFS which was a very high level description of the functional specification for the system was a good solution. The details that did not affect the scope or cost were left out at this point. The OFS provided an overview of the work tasks, services and screenshots that was to be included in the upcoming project. This worked with great success in iteration 2. Before each sprint started all of the details for the sprint were put together in a document called the DFS, which included the IT-services and the functional details that was to be provided during the sprint. The DFS provided the last 20% of the details and included all the screenshots that were supposed to be made during the sprint. In addition, the DFS included a detailed description of all the functionalities provided by the screenshots. All the necessary details were provided by this document to make sure that the developer had to do nothing else than develop according to the description.

The experiences from this project was that user stories supplemented with the IAF framework, which include the use of IT-services, work tasks and screenshots helped

make the specification end up at an appropriate level of detail before the development started.

One might think this is too rigid, but up until the sprint started, they had the opportunity to change functionality and come with potential changes or objections. This results in a flexible solution for both parties.

It was also pointed out that is important to know when to stop, when the specification is at a sufficient detail level. One can try to detail every possible outcome, but this will only cost time, money and might lead to a planning paradox.

6.2.2 Why it has worked

Several reasons were pointed out for why the chosen solution has worked as well as it has:

- Good cooperation between vendor and customer
    - Frequent meetings when making OFS and DFS
- Delay last 20% of details for the DFS
- Specify only what is needed to determine the scope of the project in the OFS and wait with the rest of the details for the DFS

Due to the project size, complexity and importance, they have put huge effort into making a specification for the project that both the customer and the vendor were happy with. Because of the size, complexity and importance of the system, it was put more effort into documentation than usually wanted in an agile project. They saw early that a detailed description was needed for the case.

The customer was also very helpful during the whole process and the cooperation between the consultants and the customer was one of the main reasons why this has worked. If the customer is not one who is able to cooperate with the developers during the project course, the approach that has been used here will not work and one should choose another approach. This is because a huge amount of cooperation from both parties were required.

For a different type of project, where there is not an outside party involved in the development, the size is smaller and the complexity is lower, you might not need to put this much effort into the specification.

The interviewees dedicate a lot of the honor of the project going so well to the DFS and because they have such a thorough requirements specification in the project.

6.2.3 Negative

There was pointed out some negative aspects with the chosen method in the case. This was amongst other that the DFS, due to the amount of detail, got very big. The amount

of detail also resulted in the developers not having room for interpretation and finding possible better solutions during the Scrum sprint. These aspects contradicts the agile manifesto which amongst other focus on little documentation.

6.3 Question 3

RQ3: When are user stories sufficient and when are other techniques or tools required?

Based on the experiences from the project one can say that for big and complex projects that is not an in-house project. User stories in itself might not be covering enough. We need something more. Something to supplement the stories and manage them in a good way.

In the case I have been studying they decided to use a more visual and more thorough description than what user stories on its own provide. They made something called a DFS to originally supplement the user stories. However, it ended up with almost replacing the user stories all together in the development. User stories were not used at all by the developers.

6.3.1 Experiences with user stories in the case

From the project, it was seen that it was started with specifying the requirements as user stories where they made a long list of stories that made up the user requirements of the project. The user stories were more or less kept during the whole project, but they were supplemented with a more detailed description, which resulted in the user stories not being used in development at all. What was also seen was that it was decided to move away from using just user stories. It was discovered that user stories on its own were:

- Not covering enough: not possible to see the whole picture
- Difficult for the customer when they wanted to change functionalities: loads of technicalities
- Resulted in a big unmanageable list of stories
- Hard to navigate through
- Hard to see connections between the stories

This resulted in:

- User stories being used only as a guide through the DFS document
- IT-services more important than user stories
- User stories being made from screenshots not the other way
- DFS worked as user stories and was what the developers worked from in the sprint
- Developers did not use user stories in the sprint

It was seen that user stories on its own were not good enough for such a big and complex project.

However, there were experienced some difficulties due to not using user stories. When testing, instead of writing a test case for a user story, the testers had to write tests for a whole chapter in the DFS. This was somewhat difficult due to the size of the chapter and that the test system was not rigged for that type of test, but this problem was something they managed to navigate around.

6.3.2 In other projects

From the interviews, I also got some opinions on user stories in general. According to some of the interviewees, user stories can be enough if the project is small enough or the project is for example an in-house project. When the project is big and complex one require more detail than in a small project. This does not necessarily exclude the use of user stories, but might require something more.

6.4 Recommendations

There are some recommendations I will make based on what has been discovered during the case study.

The use of user stories is not always enough for a big project even though an agile development methodology is used. The experiences from the case that has been studied was that user stories can be used as an initial mapping of requirements, but this is not enough. User stories are difficult to manage when there are too many of them and they are difficult to connect. Therefore, one can use something similar to the OFS and DFS.

Another recommendation is to wait with the final details until right before development. In the studied case, this has been pointed out as a reason for fewer big changes in the project.

7 Limitations of the study

Limitations of the study are discussed and somewhat justified here.

7.1.1 The project is not totally agile

This project was not completely agile. The vendor did not claim that they were completely agile either. However, many of the elements from scrum were used during development. In addition, in the time up until development they were very flexible when it came to adding and removing functionalities, but they were rigid when it came to creating a detailed documentation before they started. This was due to external factors like a customer that required a certain system to be delivered.

Therefore, I cannot claim that these findings are transferable to all agile projects, but for similar big, complex, important and semi agile projects that require some more detailed documentation due to for example an important customer. These results would probably apply.

7.1.2 Only one case

I have only been able to study one case. Studying only one case might lead to less reliable results than if more cases had been studied. However, this case has been interesting because they introduced many interesting techniques that were made for this particular project. It was also a somewhat unnormal case due to requirements not being specified by the customer before the vendor came into the picture. This uniqueness makes the case somewhat difficult to compare to other cases. However, the results that were found by studying this case might still be applicable to other cases. In addition, the suggested methods could work well for similar projects.

As mentioned earlier it is not recommended to lock in on one case too early. This is due to risks involved when having only one case. It is not necessary that the case turn out as you thought it would. After a while, you might see that the questions are not relevant for the case you have chosen or there might be difficulties with continuing to study the case. Luckily, this did not happen in the case I chose for my thesis.

7.1.3 Few interviews

More interviews could have supported the findings even more, but there was somewhat consistency to the answers from the interviewees. Therefore, one can say that there was a sufficient number of interviews.

The company in question was very open and supportive when helping me investigate the case. Therefore, I do not think it would have been a problem to get more interviews, but time was needed to write down and analyze the results.

7.1.4 Uncertainties around interview opinions

There were still some uncertainties regarding some of the opinions of the interviewees. The viewpoints of the interviewees may be hard to interpret. Some of the interviewees did not always express themselves in a way that was understandable. Moreover, they sometimes did not conclude their arguments. Ideally, I would have held a follow-up interview to clarify vagueness and uncertainties that were discovered when analyzing the interviews. Unfortunately, I did not have the capacity to do this in the boundaries of a master thesis.

Moreover, there were some uncertainties when it came to what was meant by sufficient detail level. This was a bit hard to extract from the interviews.

7.1.5 Interviewer with no previous experience

Another aspect that is important to point out is that I did not have previous experience with doing interviews. Therefore, the interviewing was a somewhat trial and error approach where something more was learnt after each interview that was held.

8 Conclusion

I found some interesting, but not so unexpected results during my study of the case. The answers I found to my research questions are summarized here.

RQ1: How thorough should the requirements specification be before development starts?

There are somewhat different experiences when it comes to this. Most of the interviewees agreed that the more details before development starts, the better. One of the interviewees on the other hand, thought that this was very project dependent and that they might have too many details defined in the DFSs, which make the DFSs too big and take away the developers opportunity to use their imagination.

Most of the interviewees were pleased with the level of details that were used in this project although the functional architect DFS and scrum master would have liked the specification to be a little bit more detailed. The DFS could have been supplement with process drawings and a more detailed description of possible error situations.

Another important aspect that seems to have worked for this case is that they made a 80% description of the functionalities in the OFS before the project started and then the remaining 20% right before the sprint started. When development started every detail was explained in the DFS so that the developers had to do nothing but develop. This saved them time and money due to fewer mistakes and fewer questions being asked during development.

RQ2: How can one make the specification thorough enough?

From the case I have studied, I have learned that one can make the requirements description thorough enough by specifying and finding the requirements in cooperation with the customer, like done in the case I have been studying. They did this through a series of meetings where the customer got an opportunity to review their wishes over and over again. The final details which did not affect the cost or scope did not have to be determined before the point where the given functionality was to be included in the sprint.

The description that was being developed from was the DFS, which included a very detailed description of; work tasks, IT-services, screenshots and a detailed description of the all the functionalities the screenshots provided. This description proved very helpful in this project and was pointed out as one of the main reasons for the projects success. Some dissatisfaction with the size of this document due to the high amount of details was nevertheless uttered, but the interviewees were very pleased with how the project was going at the time of the interviews.

RQ3: When are user stories sufficient and when are other techniques or tools required?

User stories are not always good enough. Even though the development method is agile, the use of only user stories depends on the size and scope of the project and is often better suited for small, in-house projects. If the project is small and very agile, user stories by its own might be enough. This claim is based on the interviewees opinions. When the project is large and a customer requires that something specific needs to be delivered within a contract, user stories might not be a good enough description. When this is the case, something more, to at least supplement the user stories are needed. Something to make them more manageable. The method chosen in this particular case might not be the best one, but something that has proven to work for this project. The chosen method was to apply the architecture framework IAF and create an OFS and DFS, where user stories were slightly put aside.

One can conclude that this study shows that user stories are not always good enough in big and complex project. In addition, for example, using a DFS or something similar is necessary to get the correct level of detail in the specification before one starts developing for a big and important project like this. This has worked well in the project that was studied.

9 Future work

Future work on this subject could be to do a multiple case study on different size projects and in different companies to see if they encounter the same problems that have been encountered here. It would also be interesting to find out how they have chosen to solve the problems and possibly try to introduce the same method that was used here on other projects to see if they work equally well on other projects in for example other companies.

During the study I have focused mostly on the functional aspects of the requirements. This is because it was easier to come into contact with the relevant people when it came to functional requirements. This was also something that was easier to study. During the study I have seen how they work with the functional specification and seen documentation of the functional aspects of the system. I would have liked to also see how they deal with the non-functional aspects. I have gotten some insight from the interviews, but have not been able to attend any meetings or see any documentation about the non-functional requirements. This is something that requires further investigation and would have been interesting to look into.

It would also have been interesting to go into greater detail of what each of the different roles meant about each subject. And possibly study if there is a link between the role of the interviewees and what they meant about the different subjects.

10 Bibliography

1. Somerwille, I., *Software Engineering*. Eight edition ed. 2007: Addison-Wesley.

2. Paetsch, F., A. Eberlein, and F. Maurer. *Requirements engineering and agile software development*. in *Enabling Technologies: Infrastructure for Collaborative Enterprises, 2003. WET ICE 2003. Proceedings. Twelfth IEEE International Workshops on*. 2003.

3. Beck, K., *Extreme Programming explained*. 1999: Addison-Wesley.

4. Scrum [cited 2014 13 march]; Available from: http://epf.eclipse.org/wikis/scrum/.

5. James Cadle, D.Y., *Project management for information systems*. fifth edition ed. 2008.

6. Blankenship, J., M. Bussa, and S. Millett, *Managing Agile Projects with Scrum*, in *Pro Agile .NET Development with Scrum*. 2011, Apress. p. 13-27.

7. HIRANABE, K. *Kanban Applied to Software Development: from Agile to Lean*. 2008 [cited 2014 10 january]; Available from: http://www.infoq.com/articles/hiranabe-lean-agile-kanban.

8. Olausson, M., et al., *Kanban*, in *Pro Team Foundation Service*. 2013, Apress. p. 91-99.

9. Sillitti, A. and G. Succi, *Requirements Engineering for Agile Methods*, in *Engineering and Managing Software Requirements*, A. Aurum and C. Wohlin, Editors. 2005, Springer Berlin Heidelberg. p. 309-326.

10. Robert C. Martin; Steve Mellor; Ken Schwaber; Jeff Sutherland; Dave Thomas, K.B.M.B.A.v.B.A.C.W.C.M.F.J.G.J.H.A.H.R.J.J.K.B.M. *Manifesto for Agile Software Development*. 2001 [cited 2013 4 november]; Available from: http://agilemanifesto.org/.

11. *Scrum*. 2013; Available from: http://epf.eclipse.org/wikis/scrum/Scrum/guidances/supportingmaterials/resources/ScrumLargeLabelled.png.

12. Poppendieck, M. and M.A. Cusumano, *Lean Software Development: A Tutorial.* Software, IEEE, 2012. **29**(5): p. 26-32.

13. *Kanban board*. [cited 2013 11 oct]; Available from: http://billigereogbedre.files.wordpress.com/2010/05/kanbantavle1.png?w=300&h=141.

14. Adikari, S., C. McDonald, and J. Campbell, *Little Design Up-Front: A Design Science Approach to Integrating Usability into Agile Requirements Engineering*, in *Human-Computer Interaction. New Trends*, J. Jacko, Editor. 2009, Springer Berlin Heidelberg. p. 549-558.

15. Ramesh, B., L. Cao, and R. Baskerville, *Agile requirements engineering practices and challenges: an empirical study.* Information Systems Journal, 2010. **20**(5): p. 449-480.

16. Waldmann, B. *There's never enough time: Doing requirements under resource constraints, and what requirements engineering can learn from agile development*. in *Requirements Engineering Conference (RE), 2011 19th IEEE International*. 2011.

17. Lopez-Nores, M., et al. *An agile approach to support incremental development of requirements specifications*. in *Software Engineering Conference, 2006. Australian*. 2006.

18. Leffingwell, D., *Agile Software Requirements: Lean Requirements Practices for Teams, Programs, and the Enterprise*. 1 edition ed. Agile Software Development Series. 2011: Addison-Wesley Professional.

19. Savolainen, J., J. Kuusela, and A. Vilavaara. *Transition to Agile Development - Rediscovery of Important Requirements Engineering Practices*. in *Requirements Engineering Conference (RE), 2010 18th IEEE International*. 2010.

20. Maierhofer, S., et al., *Requirement Changes and Project Success: The Moderating Effects of Agile Approaches in System Engineering Projects*, in *Systems, Software and Services Process Improvement*, A. Riel, et al., Editors. 2010, Springer Berlin Heidelberg. p. 60-70.

21. Bjarnason, E., et al., *A case study on benefits and side-effects of agile practices in large-scale requirements engineering*, in *Proceedings of the 1st Workshop on Agile Requirements Engineering*. 2011, ACM: Lancaster, United Kingdom. p. 1-5.

22. Lan, C. and B. Ramesh, *Agile Requirements Engineering Practices: An Empirical Study.* Software, IEEE, 2008. **25**(1): p. 60-67.

23. Liu, J., Q. Wang, and L. Gao. *Application of Agile Requirement Engineering in Modest-Sized Information Systems Development*. in *Software Engineering (WCSE), 2010 Second World Congress on*. 2010.

24. Yin, R.K., *Case study research: Design and Methods.* 2008.

25. Münch, J., et al., *Software Process Definition and Management*. 2012: Springer.

26. Becker-Kornstaedt, U., *Towards systematic knowledge elicitation for descriptive software process modeling*, in *Product Focused Software Process Improvement*. 2001, Springer. p. 312-325.

27. Jack van ’t Wout, M.W., Herman Hartman, Max Stahlecker, Aaldert Hofman, *The Integrated Architecture Framework Explained*. 2010: Springer.